

DeepSec: Deciding Equivalence Properties for Security Protocols — Improved theory and practice

Received Nov 14, 2022
 Revised Dec 12, 2023
 Accepted Jan 28, 2024
 Published Mar 12, 2024

Key words and phrases
 Verification, cryptographic protocols, process equivalences

Vincent Cheval^a ✉ 

Steve Kremer^b ✉ 

Itsaka Rakotonirina^c ✉ 

^a University of Oxford, United Kingdom

^b Inria Centre at Université de Lorraine, France

^c MPI-SP, Germany

ABSTRACT. Automated verification has become an essential part in the security evaluation of cryptographic protocols. In this context privacy-type properties are often modelled by indistinguishability statements, expressed as behavioural equivalences in a process calculus. In this article we contribute both to the theory and practice of this verification problem. We establish new complexity results for static equivalence, trace equivalence and labelled bisimilarity and provide a decision procedure for these equivalences in the case of a bounded number of protocol sessions. Our procedure is the first to decide trace equivalence and labelled bisimilarity exactly for a large variety of cryptographic primitives—those that can be represented by a subterm convergent destructor rewrite system. We also implemented the procedure in a new tool, DEEPSEC. We showed through extensive experiments that it is significantly more efficient than other similar tools, while at the same time raising the scope of the protocols that can be analysed.

1. Introduction

The use of automated, formal methods has become indispensable for analysing complex security protocols, such as those for authentication, key exchange and secure channel establishment. Nowadays there exist mature, fully automated such analysers; among others AVISPA [9], PROVERIF [19], SCYTHYER [46], TAMARIN [54] or Maude-NPA [58]. These tools are able to automatically verify full fledged models of widely deployed protocols and standards, such as

A preliminary version of this work appeared at IEEE Symposium on Security and Privacy (S&P) 2018 [31]. This work has been partly supported by the ANR Research and teaching chair in AI ASAP (ANR-20-CHIA-0024) with support from the region Grand Est and ANR France 2030 project SVP (ANR-22-PECY-0006).

the TLS protocol for secure connexion [17, 45], the Signal messaging protocol [53, 40], authentication protocols of the 5G standard [12], or deployed multi-factor authentication protocols [52]. Theory-wise, the tools operate in so-called *symbolic* models, rooted in the seminal work by Dolev and Yao [48]: the attacker has full control over the communication network, unbounded computational power, but cryptography is idealised. This model is well suited for finding attacks in the protocol logic, and tools have indeed been extremely effective in discovering this kind of flaw or proving their absence.

While most works investigate *reachability* properties, a later trend consists in adapting the tools—and the underlying theory—to the more complex *indistinguishability* properties. Such properties are generally modelled as a behavioural equivalence (bisimulation or trace equivalence) in a dedicated process calculus such as the spi calculus [4] or the applied pi calculus [1]. A typical example is real-or-random secrecy: after interacting with a protocol, an adversary is unable to distinguish the *real* secret used in the protocol from a *random* value. Privacy-type properties can also be expressed as such: anonymity may be modelled as the adversary’s inability to distinguish two instances of a protocol executed by different agents; vote privacy [47] has been expressed as indistinguishability of the situations where the votes of two agents have been swapped or not; unlinkability [6] is seen as indistinguishability of two sessions, either both executed by the same agent A , or by two different agents A and B .

Contributions

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

1. tight complexity results for several equivalence relations: static equivalence, trace equivalence and labelled bisimilarity. In addition to the conference paper [31], we showcase the generality of our approach by providing, with a negligible proof overhead, a tight analysis of other security relations, namely similarity, simulation, and trace inclusion;
2. a novel procedure deciding all of the above mentioned security relations for a bounded number of sessions, for the class of cryptographic primitives modelled by a destructor subterm convergent rewrite system;
3. an implementation of our procedure for trace equivalence in a tool called DEEPSEC (Deciding Equivalence Properties for SECURITY protocols), improved compared to its initial presentation in the conference paper [31].

We detail the three contributions below.

Complexity We provide the first complexity results for deciding trace equivalence and labelled bisimilarity in the applied pi calculus, without any syntactic or semantic restriction on the class of protocols (other than bounding the number of sessions), and for a large class of cryptographic primitives modelled as rewrite rules. As mentioned above, our results extend to several other security relations such as simulation. Let us also highlight one small, yet substantial difference with existing work: we do not consider cryptographic primitives (rewrite systems) as constants of the problem. As most modern verification tools allow for user-specified primitives [22, 54, 58, 24], our approach seems to better fit this reality. Typically, all existing procedures for static equivalence can only be claimed PTIME because of this difference and are actually exponential in the sizes of the signature or equational theory. Our complexity results are summarised in Table 1. All our lower bounds hold for subterm convergent rewrite systems and even for the positive fragment (without else branches). *En passant*, we present results for the pi calculus: although investigated in [23], complexity was unknown when restricted to a bounded number of sessions. Still, our main result is the coNEXP completeness (and in particular, the decidability) of trace equivalence and labelled bisimilarity for destructor subterm convergent rewrite systems.

	Pure pi calculus	Applied pi calculus with destr. subterm convergent theory
static equivalence	LOGSPACE	coNP complete
trace equivalence	Π_2 complete	coNEXP complete
labelled (bi)similarity	PSPACE complete	coNEXP complete

Table 1. Summary of complexity results

Decision procedure We present a novel procedure based on a symbolic semantics and constraint solving. Unlike most other work, our procedure decides equivalences *exactly*, i.e. without approximations. Moreover, it does not restrict the class of processes (except for replication), nor the use of else branches, and is correct for any cryptographic primitives that can be modelled by a subterm convergent destructor rewrite system (see Section 2 for more details). The design of the procedure did greatly benefit from our complexity study, and was developed in order to obtain tight complexity upper bounds. The theory is also more mature compared to the initial conference paper [31] which allowed some significant optimisations of the constraint solving procedure.

Tool implementation We implemented our procedure for trace equivalence in a tool, DEEPSEC. Its prototype has initially been presented in the conference paper [31] (and some implementation details in a tool paper [33]), but has significantly matured since then. In addition of the improvements at the level of the theoretical procedure, the low level implementation has been

more carefully engineered data-structure-wise. All in all, the DEEPSEC 2.0.0 release includes the following new features:

- A significantly *reduced verification time* (several orders of magnitude on some examples).
- An *optional procedure exploiting the symmetries* that often arise in practical verification. When used, this further reduces the verification time by orders of magnitude albeit for occasionally introducing false attacks. In this article we rather focus on the main procedure; details about this feature can be found in [32].
- An *improved user experience*. The html based pretty-print of the original prototype has been upgraded into a standalone graphical user interface. Verification queries and options can be managed directly from the interface and a simulator displays interactively equivalence proofs or attacks to better visualise the outcome of the analysis.

Naturally DEEPSEC still integrates already-present features such as multicore distribution and the partial order reductions presented in [10]. All in all this makes the tool more user friendly and scale well despite the high theoretical complexity of the problem (coNEXP). Installation guidelines can be found in the official website [35] together with a manual and a tutorial.

Through extensive benchmarks, we compare DEEPSEC to other tools limited to a bounded number of protocol sessions: APTE, SPEC, AKISS, SATEQUIV and our previous prototype (as presented in [31]). This prior version was already more efficient—by several orders of magnitude—than APTE, SPEC and AKISS, even though DEEPSEC covers a strictly larger class of protocols than APTE and SPEC. Besides, its performances were comparable to SATEQUIV, which still outperforms DEEPSEC when the number of parallel processes significantly increase. This gap in performance seems unavoidable as DEEPSEC operates on a much larger class of protocols (more primitives, else branches, no limitation to simple processes, termination guaranteed). Part of the benchmarks consists of classical authentication protocols and focuses on demonstrating scalability of the tool when augmenting the number of parallel protocol sessions. The other examples include more complex protocols, such as Abadi and Fournet’s anonymous authentication protocol [3], the protocols implemented in the European passport [51], a model (without XOR) of the AKA protocol used in 3G mobile telephony, as well as the Prêt-à-Voter [57] and the Helios [5] e-voting protocols.

Related Work

The problem of analysing security protocols is undecidable in general but several decidable subclasses have been identified. While many complexity results are known for trace properties [49, 56], the case of behavioural equivalences remains mostly open. When the attacker is an eavesdropper and cannot interact with the protocol, the indistinguishability problem—*static equivalence*—has been shown PTIME for large classes of cryptographic primitives [2, 39, 42]. For active attackers, bounding the number of protocol sessions is often sufficient to

obtain decidability [56] and is of practical interest: most real-life attacks indeed only require a small number of sessions. In this context Baudet [14], and later Chevalier and Rusinowitch [36], showed that real-or-random secrecy was coNP for cryptographic primitives that can be modelled as subterm convergent rewrite systems, by checking whether two constraint systems admit the same set of solutions. These procedures do however not allow for else branches, nor do they verify trace equivalence in full generality. In [29], Cheval et al. have used Baudet's procedure as a black box to verify trace equivalence of *determinate* processes. This class of processes is however insufficient for most anonymity properties. Finally, decidability results for an unbounded number of sessions exist [38, 37], but with severe restrictions on processes and equational theories.

Tool support also exists for verifying equivalence properties. We start discussing tools that are limited to a bounded number of sessions. The SPEC tool [59, 60] verifies a sound symbolic bisimulation, but is restricted to particular cryptographic primitives (pairing, encryption, signatures and hash functions) and does not allow for else branches. In a similar setting, restricting to particular primitives, Cheval et al. [27] propose a procedure for deciding equivalence of constraint systems. This procedure can be used for deciding trace equivalence of determinate processes and has been implemented in the ADECS tool. The APTE tool [25] generalizes ADECS: it covers the same primitives but allows else branches and decides trace equivalence exactly. On the contrary, the AKISS tool [24] allows for user-defined cryptographic primitives. The procedure of this tool is correct for primitives modelled by an arbitrary convergent rewrite system that has the finite variant property [41], and termination is additionally guaranteed for subterm convergent rewrite systems. However, AKISS does only decide trace equivalence for a class of determinate processes; for other processes trace equivalence can be both over- and under-approximated. The recent SATEQUIV tool [43] uses a different approach: it relies on Graph Planning and SAT solving to verify trace equivalence, rather than a dedicated procedure. The tool is extremely efficient and several orders of magnitude faster than other tools. It does however not guarantee termination and is currently restricted to pairing and symmetric encryption and only considers a class of *simple processes* (a subclass of determinate processes) that satisfy a type-compliance condition. These restrictions severely limit its scope.

To mitigate the state explosion problem from which most of the above tools suffer, Baelde et al. [10] developed *partial order techniques* which avoid to explicitly consider all possible interleavings and which are compatible with a symbolic approach based on constrained solving. Substantial efficiency gains on practical examples have been illustrated through an implementation in the APTE tool. We also implemented these techniques in DEEPSEC. However, the techniques may only be applied on a class of *action-determinate* processes. This limitation has been overcome in a follow-up work by Baelde et al. [12]: while more general the new techniques also require additional, expensive computations resulting in less spectacular performance in-

crease than the initial work. Baelde et al. [12]. have implemented their technique in a standalone library, and plugged it into the APTE and DEEPSEC tools.

Other tools support verification of equivalence properties, even for an unbounded number of sessions. This is the case of PROVERIF [20], TAMARIN [13] and Maude NPA [58] which all allow for user-defined cryptographic primitives. However, given that the underlying problem is undecidable, these tools may not terminate. Moreover, they only approximate trace equivalence by verifying the stronger *diff-equivalence*. This equivalence is too strong on many examples. While some recent improvements on PROVERIF [26, 21] help covering more protocols, general verification of trace equivalence is still out of scope. For instance, the verification by Arapinis et al. [8] of unlinkability in the 3G mobile phone protocols required some “tricks” and approximations of the protocol to avoid false attacks. In [44], Cortier et al. develop a type system and automated type checker for verifying equivalences. While extremely efficient, this tool only covers a fixed set of cryptographic primitives (the same as SPEC and APTE) and verifies an approximated equivalence, similar to *diff-equivalence*. A different approach has been taken by Hirschi et al. [50], identifying sufficient conditions provable by PROVERIF for verifying unlinkability properties, implemented in the tool UKANO, a front-end to the PROVERIF tool. UKANO does however not verify equivalence properties in general.

Article Outline

We organize the article as follows. In Section 2 we present our formal model of cryptographic protocols and the process equivalences used to express security properties. We also precisely define the decision problems that we address in this article.

In Section 3 we provide an overview of our decision procedures. First, we define (sound and complete) symbolic semantics where we replace the infinite set of possible attacker inputs by a finite representation in the form of constraint systems. Second, we define the notion of a *partition tree*. The partition tree organizes all symbolic traces in a tree such that a node contains (a symbolic representation of) all statically equivalent processes that can be reached by a given trace. Third, we show how equivalences can be decided on such a partition tree. Next, we explain how to compute a partition tree *assuming* we can compute solutions to constraint systems. Finally, we discuss how the procedure for deciding trace equivalence has been implemented in the DEEPSEC tool and provide a performance evaluation.

In Section 4 we present a rule-based procedure to effectively solve constraint systems. This requires the definition of *extended* constraint systems that store additional information and the introduction of the new notion of *most general solutions*. Reminiscent of the notion of most general unifiers, most general solutions are a set of solutions that guarantee that any solution can be obtained from a most general solution by substituting atomic names by more complex terms. After presenting all the rules of the procedure in detail we explain how to construct a partition tree.

In Section 5 we give complexity results. To achieve upper bounds we prove termination of the constraint solving procedure and exponentially bound the number of rules and size. From these bounds we obtain that when two processes are not equivalent (for different notions of equivalence) there exists a witness of exponential size, yielding a coNEXP decision procedure for equivalence. Lower bounds are provided by reduction to the `SUCCINCTSAT` problem.

Finally we conclude the article in Section 6 and sketch some directions for future work.

2. Model

We first present our model of cryptographic protocols and use it to model the security of the Private Authentication Protocol as a running example [3]. Our framework is based on the applied pi calculus [1] and follows the tradition of symbolic models rooted in the seminal work of Dolev and Yao [48]. In these models, the low-level details of cryptography are abstracted by a term algebra describing the ideal behaviour of cryptographic primitives, whereas secret data such as cryptographic keys or nonces are represented by symbolic values called *names*.

2.1 Messages and cryptography

Protocol messages Cryptographic operations are modelled by a set \mathcal{F} of symbols of fixed arity denoted $\mathcal{F} = \{f/n, g/m, \dots\}$, called a *signature*. In this article, it is always partitioned into:

- The infinite set of *constants* (\mathcal{F}_0) that are the functions of arity 0 of \mathcal{F} , thus modelling the public values of the protocol such as identities, IP addresses or public communication channels.
- The finite set of *constructors* (\mathcal{F}_c) modelling cryptographic operations used to build messages, typically encryption, signature, concatenation or hash.
- The finite set of *destructors* (\mathcal{F}_d) modelling inversions or operations that may fail depending on the structure of their argument, typically decryption, signature verification or projection.

EXAMPLE 2.1. The following signature captures most of the cryptographic primitives that are used in our examples and benchmarks. We will use them throughout Section 2 in examples.

	concatenation / pairs	symmetric encryption	asymmetric encryption	digital signature	one-way hash
\mathcal{F}_c	$\langle \cdot, \cdot \rangle / 2$	senc/3	pk/1, aenc/3	vpc/1, sign/3	h/1
\mathcal{F}_d	fst/1, snd/1	sdec/2	adec/2	verify/2	\emptyset

For example $\text{aenc}(m, r, \text{pk}(k))$ models a plaintext m encrypted with public key $\text{pk}(k)$ and a randomness r . The corresponding decryption key would be k . A similar description can be made for symmetric encryption, except that the encryption and decryption keys are identical.

The model of hash functions contains no destructors on purpose, thus modelling an assumption that h is a random oracle, i.e., no identities can be derived from h . Notation-wise, we also often use a tuple notation $\langle x_1, \dots, x_n \rangle$ instead of the nested $n - 1$ pairs $\langle x_1, \langle x_2, \dots, \langle x_{n-1}, x_n \rangle \rangle \rangle$. \blacklozenge

A protocol message m is then modelled by a *term* over this signature, i.e. m is obtained by applying function symbols to other terms or *names*. The infinite set of *names* \mathcal{N} can be seen as a symbolic abstraction of private values such as encryption keys or nonces. The set of names occurring in a term t is written $names(t)$. In some models names are partitioned into public and private names, where the set of public names essentially plays the same role as \mathcal{F}_0 . Since constants and public names have a similar role (and are even treated identically in our tool implementation) we decided to merge them into the single set \mathcal{F}_0 similarly to other formalisations, e.g. [37]. We write $\mathcal{T}(S)$, $S \subseteq \mathcal{F} \cup \mathcal{N}$, the set of terms built from functions, names, and constants of S .

Specifying cryptographic assumptions The behaviour of the primitives of the signature is modelled by a *rewriting system*. For that we assume an infinite set of *variables* $\mathcal{X} = \{x, y, z, \dots\}$ that may be used in terms, and write $vars(t)$ the set of variables occurring in a term t . Mappings σ from variables to terms are called *substitutions* and are homomorphically extended to mappings from terms to terms implicitly. We use the postfix notation $t\sigma$ for $\sigma(t)$, and $\sigma\sigma'$ for the composition of substitution $\sigma' \circ \sigma$ (that is, $t\sigma\sigma' = (t\sigma)\sigma'$). We call the *domain* of σ the set $dom(\sigma) = \{x \in \mathcal{X} \mid x\sigma \neq x\}$. For convenience we also use set notations, defining a substitution σ such that $dom(\sigma) \subseteq \{x_1, \dots, x_n\}$ with the notation $\sigma = \{x_1 \mapsto \sigma(x_1), \dots, x_n \mapsto \sigma(x_n)\}$. Going further we may refer to the substitution $\sigma \cup \sigma'$ (provided σ and σ' coincide on $dom(\sigma) \cap dom(\sigma')$) or write $\sigma \subseteq \sigma'$ to mean that σ' extends σ . A *rewriting system* \mathcal{R} is then a finite binary relation on terms. All pairs of \mathcal{R} are called *rewrite rules* and are assumed to be of the form

$$f(\ell_1, \dots, \ell_n) \rightarrow r \quad \text{for some } f/n \in \mathcal{F}_d \text{ and } \ell_1, \dots, \ell_n, r \in \mathcal{T}(\mathcal{F}_c \cup \mathcal{F}_0 \cup \mathcal{X})$$

Such rewriting systems are usually qualified as *constructor destructor* in the literature. By extension we also use notation $t \rightarrow s$ (“ t rewrites to s ”) when t and s are related by the closure of \mathcal{R} under application of substitution and term context. The reflexive transitive closure of this relation is written \rightarrow^* .

EXAMPLE 2.2. We give the rewrite rules for the primitives introduced in Example 2.1.

<i>sym. encryption:</i>	$sdec(senc(x, y, z), z) \rightarrow x$
<i>pairs:</i>	$fst(\langle x, y \rangle) \rightarrow x \quad \text{and} \quad snd(\langle x, y \rangle) \rightarrow y$
<i>asym. encryption:</i>	$adec(aenc(x, y, pk(z)), z) \rightarrow x$
<i>signatures:</i>	$verify(sign(x, y, z), vpk(z)) \rightarrow x$

For example here one can decrypt (apply adec) a ciphertext $\text{aenc}(x, y, \text{pk}(z))$ with the corresponding key z to recover the plaintext x . The rule for signature verification is the opposite, recovering the signed message x using the public verification key $\text{vpk}(z)$. The behaviour of these primitives is idealised by the absence of other rules, for example modelling an assumption that no information can be extracted from a ciphertext or a signature without access to the secret or verification keys. This idealisation can be partially lifted by adding more rewrite rules modelling specific imperfections of the cryptography. For example we can add the following new symbols and rewrite rules:

$$\text{test_aenc}(\text{aenc}(x, y, \text{pk}(z))) \rightarrow \text{ok} \quad \text{get_key}(\text{aenc}(x, y, \text{pk}(z))) \rightarrow \text{pk}(z)$$

model two assumptions that 1. it is possible to distinguish a correctly encrypted message from a random bitstring, and 2. it is possible to retrieve the encryption key from the ciphertext itself (i.e. the scheme is not *key concealing*). Naturally even if a protocol is considered secure without these two rewrite rules, a security violation may arise upon adding them. It is therefore important to keep in mind the assumptions underlying the model when interpreting the result of an analysis. ◆

We observe that the rewrite rules introduced in the example above verify a classical property, *subterm convergence*, introduced in [2] and benefiting from several decidability results in the context of protocol analysis [2, 29]. It means that \mathcal{R} is convergent (i.e. confluent and strongly terminating) and that its rules $\ell \rightarrow r$ verify that r is either a strict subterm of ℓ or a *ground term* (i.e. a term without variables) in *normal form* (i.e. irreducible w.r.t. \rightarrow). The results of this article only apply to cryptographic primitives modelled by a constructor destructor subterm convergent rewriting systems. Imposing such restrictions is inevitable when aiming for decidability, since the problems we investigate are undecidable for arbitrary convergent rewriting systems [2].

In particular, by convergence, all terms $t \in \mathcal{T}(\mathcal{F} \cup \mathcal{N})$ have a unique normal form w.r.t. \mathcal{R} that we will write $t \downarrow$. It is also common to identify messages whose destructors failed to be applied. For that we define a predicate msg on terms: we say that t is a *message*, written $\text{msg}(t)$, when for all subterms u of t , $u \downarrow$ does not contain any destructors. For example if $m, r \in \mathcal{F}_0$ and $k \neq k'$, $\text{adec}(\text{aenc}(m, r, \text{pk}(k)), k)$ is a message but not $\text{fst}(\langle m, t \rangle)$ with $t = \text{sdec}(\text{senc}(m, r, k), k')$.

2.2 Protocols

Processes Security protocols are modelled by (*plain*) processes in a concurrent process calculus defined by the following grammar:

$P, Q := 0$	null
$P \mid Q$	parallel
$\text{if } u = v \text{ then } P \text{ else } Q$	conditional
$\bar{u}\langle v \rangle.P$	output
$u(x).P$	input

where u, v are terms and $x \in \mathcal{X}$. Intuitively the 0 models a terminated process (and is often omitted for succinctness), a conditional $\text{if } u = v \text{ then } P \text{ else } Q$ executes either P or Q depending on whether the terms u and v are messages and have the same normal form, and $P \mid Q$ models two concurrent processes. Inter-process communications are performed with $u(x).P$ and $\bar{u}\langle v \rangle.P$ which are, respectively, inputs and outputs on a communication channel u . When u is known to the attacker, for example when it belongs to \mathcal{F}_0 , executing an output on u adds it to the adversary's knowledge, whereas an input on u is fetched from the adversary possibly forwarding a previously stored message, or computing a new message from previous outputs. Otherwise the communication is performed silently without adversarial interferences. The main difference with the calculus of [1] is the absence of replication, thus bounding the number of instructions of a process. This restriction does *not* make protocol analysis trivially decidable: although the number of instructions are finite, the number of their possible executions is not, since the attacker can fetch arbitrary messages to public inputs.

EXAMPLE 2.3. We define a process modelling the protocol for private authentication described in [3] as a running example through the article. Denoting by sk_X, pk_X the secret and public keys of an agent X , and by r_X fresh nonces, its control flow can be described as follows using an informal Alice-Bob notation:

$$\begin{array}{ll}
 X \rightarrow B : \text{aenc}(\langle N_X, pk_X \rangle, r_X, pk_B) & \\
 B \rightarrow X : \text{aenc}(\langle N_X, N_B, pk_B \rangle, r_A, pk_A) & \text{if } X = A \\
 \text{aenc}(N_B, r_B, pk_B) & \text{if the decryption fails or } X \neq A
 \end{array}$$

where N_X, N_B are two freshly generated nonces. Here the agent B accepts authentication requests from the agent A but not from other parties. Among the security goals stated in [3] are

1. *Secrecy*: At the end of a successful instance of the protocol between A and B , N_A and N_B are secrets (i.e. the attacker cannot get information about them).
2. *Anonymity*: The attacker cannot tell whether the protocol is run by A and B or other agents.

3. *Private authentication*: The attacker cannot tell whether B accepts connections from A or not.

The last two security goals explain in particular the decoy message $\text{aenc}(N_B, r_B, pk_B)$ that B sends upon decryption failure or connection refusal: thus from an outside observer there is no observable difference between the situations where B answers or not. The roles of X and B can be specified as follows in the applied pi calculus; each process takes as an argument its secret key s , the public key p of the agent it aims at communicating with, its fresh session nonces n, r and we write $t = \text{adec}(x, s)$:

$$\begin{aligned}
 X(s, p, n, r) &= \bar{c}\langle \text{aenc}(\langle n, \text{pk}(s) \rangle, r, p) \rangle. & B(s, p, n, r) &= c(x). \\
 & & & \text{if } \text{snd}(t) = p \text{ then} \\
 & & & \quad \bar{c}\langle \text{aenc}(\langle \text{fst}(t), n, \text{pk}(s) \rangle, r, p) \rangle \\
 & & & \text{else } \bar{c}\langle \text{aenc}(n, r, \text{pk}(s)) \rangle
 \end{aligned}$$

where $c \in \mathcal{F}_0$. The security goals are formalised in Section 2.3. ◆

Attacker's knowledge In the next paragraphs we formalise how processes may be executed in an active adversarial environment. The first step is to model the capabilities of the underlying attacker that spies on the communication network and actively interferes with communications. For that we refine the set of variables to $\mathcal{X} = \mathcal{X}^1 \uplus \mathcal{AX}$, thus introducing a new type of variables $\mathcal{AX} = \{\text{ax}_1, \text{ax}_2, \text{ax}_3, \dots\}$ called *axioms* that will serve as handles to make reference to attacker's observations. Concretely a term $\xi \in \mathcal{T}(\mathcal{F} \cup \mathcal{AX})$ is called a *recipe* and is intuitively an algorithm for the attacker to construct a term from their prior observations. For example upon observing the messages $\text{aenc}(m, r, \text{pk}(k))$ and k in this order, an attacker can use the recipe $\xi = \text{adec}(\text{ax}_1, \text{ax}_2)$ to retrieve m although it has not been observed directly. We observe in particular that by definition a recipe cannot contain names, modelling that they are assumed to be private and as such cannot be used directly by the adversary.

On the other hand, the variables of \mathcal{X}^1 , called *first-order variables* for distinction, stick to the initial role of variables—namely, being used as binders for protocol inputs. For this reason, we call a term $t \in \mathcal{T}(\mathcal{F} \cup \mathcal{N} \cup \mathcal{X}^1)$ a *protocol term*. However, we often more specifically consider *constructor terms* $\mathcal{T}(\mathcal{F}_c \cup \mathcal{F}_0 \cup \mathcal{N} \cup \mathcal{X}^1)$ that are protocol terms whose destructors have all been successfully computed, that is, reduced by a rewrite rule. We also write $\text{vars}^1(t) = \text{vars}(t) \cap \mathcal{X}^1$ the set of *first-order variables* of t . Using all these notions we define *extended processes*, representing a set of processes executed in parallel together with the knowledge aggregated by the attacker interacting with the protocol:

DEFINITION 2.4. An extended process is a pair $A = (\mathcal{P}, \Phi)$ with \mathcal{P} a multiset of ground processes and $\Phi = \{\text{ax}_1 \mapsto u_1, \dots, \text{ax}_n \mapsto u_n\} = \Phi(A)$ is called a *frame* that is a substitution from axioms to ground constructor terms.

Formalising the example above, if the frame $\Phi = \{ax_1 \mapsto \text{aenc}(m, r, \text{pk}(k)), ax_2 \mapsto k\}$ models the attacker's observations during the execution of a protocol, the fact that m can be retrieved with the recipe $\xi = \text{adec}(ax_1, ax_2)$ is expressed by the fact that $\text{msg}(\xi\Phi)$ and $\xi\Phi \downarrow = m$. A typical security problem is to decide, given a frame Φ and a term t , whether t is *deducible* by the attacker from Φ ; that is, whether there exists a recipe ξ such that $\text{msg}(\xi\Phi)$ and $\xi\Phi \downarrow = t \downarrow$.

Operational semantics We now formalise the semantics of processes. By manipulating extended processes this semantics carries the knowledge the attacker aggregates by spying on the communication outputs. Besides, in our constructor destructor setting we assume that the agents only send and accept meaningful messages, namely terms that verify the msg predicate. While this assumption is realistic for authenticated encryption for example, it may not hold for schemes with weaker security guarantees. In practice the semantics takes the form of a transition relation between extended processes labelled by so-called *actions*:

1. *Input actions* $\xi_c(\xi_t)$, where ξ_c and ξ_t are recipes, model an input from the attacker of a message (crafted using recipe ξ_t) on some channel (known to the attacker using recipe ξ_c)
2. *Output actions* $\bar{\xi}_c\langle ax_n \rangle$, where ξ_c is a recipe, model an output on a channel (known by the attacker using recipe ξ_c), recorded into the frame (at pointer $ax_n \in \mathcal{AX}$).
3. *Silent actions* τ that model actions that are unobservable by the attacker such as synchronous private communications or evaluation of a conditional.

We call \mathcal{A} the alphabet of actions, and transitions are of the form $A \xrightarrow{a} B$, $a \in \mathcal{A}$. The transition relation is defined by the rules given in Figure 1. More generally:

DEFINITION 2.5 (trace). We write $A \xRightarrow{w} B$ when $A \xrightarrow{a_1} \dots \xrightarrow{a_n} B$ and $w \in \mathcal{A}^*$ is the word obtained after removing the τ actions from the word $a_1 \dots a_n$, and call such a sequence of transitions a *trace*. We also write $\xRightarrow{\tau}$ for $\xrightarrow{\tau}^*$, i.e. the reflexive, transitive closure of $\xrightarrow{\tau}$.

Apart from the absence of replication, this semantics aims at being as close as possible to the original semantics of the applied pi calculus [1] although using a different formalism, as it is also the semantics used by tools such as PROVERIF.

EXAMPLE 2.6. We now illustrate how our running example can be executed in the operational semantics. We let two agents of respective secret keys $sk_A, sk_B \in \mathcal{N}$. An instance of the protocol between A and B is thus modelled, using the notations of Example 2.3, by the process $P = \bar{A} \mid \bar{B}$ where, given fresh names r_A, r_B :

$$\bar{A} = X(sk_A, \text{pk}(sk_B), N_A, r_A) \qquad \bar{B} = B(sk_B, \text{pk}(sk_A), N_B, r_B)$$

$$\begin{array}{ll}
(\{\{u(x).P\}\} \cup \mathcal{P}, \Phi) \xrightarrow{\xi_c(\xi_t)} (\{\{P\{x \mapsto \xi_t \Phi\downarrow\}\}\} \cup \mathcal{P}, \Phi) & \text{if } \text{msg}(\xi_c \Phi), \text{msg}(\xi_t \Phi), \text{msg}(u) \quad (\text{IN}) \\
& \text{and } \xi_c \Phi\downarrow = u\downarrow \\
(\{\{\bar{u}\langle v \rangle.P\}\} \cup \mathcal{P}, \Phi) \xrightarrow{\bar{\xi}_c\langle ax_n \rangle} (\{\{P\}\} \cup \mathcal{P}, \Phi \cup \{ax_n \mapsto v\downarrow\}) & \text{if } \text{msg}(\xi_c \Phi), \text{msg}(u), \text{msg}(v) \quad (\text{OUT}) \\
& \xi_c \Phi\downarrow = u\downarrow \text{ and } n = |\text{dom}(\Phi)| + 1 \\
(\{\{\bar{u}\langle v \rangle.P, u'(x).Q\}\} \cup \mathcal{P}, \Phi) \xrightarrow{\tau} (\{\{P, Q\{x \mapsto v\}\}\} \cup \mathcal{P}, \Phi) & \text{if } \text{msg}(u), \text{msg}(v), \text{msg}(u') \quad (\text{COMM}) \\
& \text{and } u\downarrow = u'\downarrow \\
(\{\{\text{if } u = v \text{ then } P \text{ else } Q\}\} \cup \mathcal{P}, \Phi) \xrightarrow{\tau} (\{\{P\}\} \cup \mathcal{P}, \Phi) & \text{if } \text{msg}(u), \text{msg}(v) \text{ and } u\downarrow = v\downarrow \quad (\text{THEN}) \\
(\{\{\text{if } u = v \text{ then } P \text{ else } Q\}\} \cup \mathcal{P}, \Phi) \xrightarrow{\tau} (\{\{Q\}\} \cup \mathcal{P}, \Phi) & \text{if } \neg \text{msg}(u), \neg \text{msg}(v) \text{ or } u\downarrow \neq v\downarrow \quad (\text{ELSE}) \\
(\{\{P \mid Q\}\} \cup \mathcal{P}, \Phi) \xrightarrow{\tau} (\{\{P, Q\}\} \cup \mathcal{P}, \Phi) & \quad (\text{PAR})
\end{array}$$

Figure 1. Semantics of the calculus

In order to lighten the presentation we use the same notations as in Example 2.3 and name the three messages of the protocol as follows:

$$\begin{aligned}
m_A &= \text{aenc}(\langle N_A, \text{pk}(sk_A) \rangle, r_A, \text{pk}(sk_B)) \\
m_B &= \text{aenc}(\langle \text{fst}(t), N_B, \text{pk}(sk_B) \rangle, r_B, \text{pk}(sk_A)) \\
m'_B &= \text{aenc}(N_B, r_B, \text{pk}(sk_B))
\end{aligned}$$

We assume that the public keys $\text{pk}(sk_A)$ and $\text{pk}(sk_B)$ are known to the attacker, which can be modelled by an initial frame $\Phi_0 = \{ax_1 \mapsto \text{pk}(sk_A), ax_2 \mapsto \text{pk}(sk_B)\}$. Another possibility is to prefix the process P with two outputs of $\text{pk}(sk_A)$ and $\text{pk}(sk_B)$ respectively, which will produce the frame Φ_0 after two applications of rule (OUT). The normal execution of the process is the following sequence of reduction steps:

$$\begin{aligned}
(\{\{P\}\}, \Phi_0) &\xrightarrow{\tau} (\{\{\bar{A}, \bar{B}\}\}, \Phi_0) \\
&\xrightarrow{\bar{c}\langle ax_3 \rangle} (\{\{c(x), \bar{B}\}\}, \Phi_1) \quad \text{with } \Phi_1 = \Phi_0 \cup \{ax_3 \mapsto m_A\} \\
&\xrightarrow{c(ax_3)} (\{\{c(x), \bar{c}\langle m_B \rangle\}\}, \Phi_1) \\
&\xrightarrow{\bar{c}\langle ax_4 \rangle} (\{\{c(x), 0\}\}, \Phi_2) \quad \text{with } \Phi_2 = \Phi_1 \cup \{ax_4 \mapsto m_B\} \\
&\xrightarrow{c(ax_4)} (\{\{0, 0\}\}, \Phi_2)
\end{aligned}$$

In this execution the attacker only forwards messages, that is, each input action uses the last axiom added to the frame as a recipe. However the adversary may actively engage in the protocol, for example for guessing whether B accepts communications from a third agent C . For that they could generate fresh nonces $N, R \in \mathcal{F}_0$ (attacker-generated nonces are modelled by fresh constants) and send the message $m'_A = \text{aenc}(\langle N, \text{pk}(sk_C) \rangle, R, \text{pk}(sk_B))$ to check how B

responds. Note that the message m'_A can indeed be crafted by the attacker assuming $\Phi'_0 = \Phi_0 \cup \{ax_3 \mapsto pk(sk_C)\}$ as an initial frame. This scenario corresponds to the following sequence of transitions:

$$\begin{aligned} (\{P\}, \Phi'_0) &\xrightarrow{\tau} (\{\bar{A}, \bar{B}\}, \Phi'_0) \\ &\xrightarrow{c(\text{aenc}(\langle N, ax_3 \rangle, R, ax_2))} (\{\bar{A}, \bar{c}\langle m'_B \rangle\}, \Phi'_0) \\ &\xrightarrow{\bar{c}\langle ax_4 \rangle} (\{\bar{A}, 0\}, \Phi'_0 \cup \{ax_4 \mapsto m'_B\}) \end{aligned}$$

This does not leak information to the attacker, assuming they cannot distinguish the messages m_B and m'_B . All in all, the set of *traces* of the process, i.e. the set of all possible sequences of reductions, characterises all possible executions of the protocol in an active adversarial environment. \blacklozenge

As a final note, let us observe that the original pi calculus [55] (referred as the *pure pi calculus* in this article) can be seen as a special case of our model. Indeed the fragment without replication is retrieved when \mathcal{F}_c , \mathcal{F}_d and \mathcal{R} are empty. This restriction makes the transition relation finitely branching up to bijective renaming of attacker-generated constants.

2.3 Security properties

Against a passive attacker We first define the notion of *static equivalence* that is often used to model security against a passive attacker in that it is only an equivalence of frames, i.e. it does not involve the operational semantics. It expresses that the knowledge obtained by eavesdropping in two different situations does not permit the attacker to distinguish them. For example no differences can be observed between $\{ax_1 \mapsto k\}$ and $\{ax_1 \mapsto k'\}$ if $k, k' \in \mathcal{N}$ because, intuitively, two fresh nonces look like random bitstrings from an external observer's point of view. However the situation is different with the frames

$$\Phi = \{ax_1 \mapsto k, ax_2 \mapsto k\} \quad \Psi = \{ax_1 \mapsto k', ax_2 \mapsto k\} \quad \text{with } k \neq k'$$

Indeed, even if no differences can be made between k and k' in isolation, the attacker observed two identical messages in the first situation but two different messages in the second situation. In particular we say that the equality test “ $ax_1 = ax_2$ ” distinguishes the two frames (because it holds in Φ but not in Ψ). Besides, in our constructor destructor algebra it is also possible to observe destructor failures. For example the following frames can be distinguished:

$$\Phi = \{ax_1 \mapsto k, ax_2 \mapsto \text{aenc}(m, r, pk(k))\} \quad \Psi = \{ax_1 \mapsto k', ax_2 \mapsto \text{aenc}(m, r, pk(k))\}$$

Indeed crafting the recipe $\text{adec}(ax_2, ax_1)$ (i.e. decrypting the last observed message with the first one) succeeds in the first situation but triggers a decryption failure in the second. Static equivalence has been extensively studied in the literature (see e.g. [2, 39, 16, 42]). Formally:

DEFINITION 2.7. Two frames Φ and Ψ of same domain are *statically equivalent*, written $\Phi \sim \Psi$, when for all recipes ξ, ζ :

1. $\text{msg}(\xi\Phi)$ if and only if $\text{msg}(\xi\Psi)$
2. assuming $\text{msg}(\xi\Phi)$ and $\text{msg}(\zeta\Phi)$, $\xi\Phi \downarrow = \zeta\Phi \downarrow$ if and only if $\xi\Psi \downarrow = \zeta\Psi \downarrow$.

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

EXAMPLE 2.8. The fact that the two frames

$$\Phi = \{ax_1 \mapsto \text{aenc}(m, r, \text{pk}(k))\} \quad \Psi = \{ax_1 \mapsto k'\} \quad \text{with } m \in \mathcal{F}_0 \text{ and } k, k', r \in \mathcal{N}$$

are statically equivalent intuitively models that encryption makes messages unintelligible (in that the attacker cannot distinguish a ciphertext from a fresh nonce). Naturally this does not hold any more once the decryption key is revealed. Formally: $\Phi \cup \{ax_2 \mapsto k\} \not\sim \Psi \cup \{ax_2 \mapsto k\}$ as witnessed by the recipe $\xi = \text{adec}(ax_1, ax_2)$ whose computation succeeds in the first frame but triggers a decryption failure in the second. Without going to the extreme extent of revealing the key, the two situations are also distinguishable if we weaken the cryptographic assumptions on aenc . For example, recalling the considerations of Example 2.1, if we do not suppose the encryption scheme to be *key concealing* anymore by adding the rule

$$\text{get_key}(\text{aenc}(x, y, \text{pk}(z))) \rightarrow \text{pk}(z)$$

then Φ and Ψ are distinguished by the recipe $\text{get_key}(ax_1)$ whose destructor succeeds in Φ but fails in Ψ . The same fact would arise using the weaker rewrite rule

$$\text{test_aenc}(\text{aenc}(x, y, \text{pk}(z))) \rightarrow \text{ok}$$

that tests whether a bitstring is a ciphertext. ◆

Against an active attacker Dynamic extensions of static equivalence consider distinguishability for an attacker interacting actively with protocols. Consider for example a protocol modelled by a process P manipulating a nonce k . A possible model of the secrecy of k can be formalised by a non-interference statement: there is no observable difference in the behaviour of the protocol when k is replaced by another term. In this article we study several relations modelling the underlying notion of indistinguishability. For completeness, we also present their associated pre-orders that can be useful modelling tools in situations where only inclusion relations are to be expressed.

DEFINITION 2.9 (Trace equivalence). If A and B are extended processes, we write $A \sqsubseteq_t B$ when for all traces $A \xRightarrow{\text{tr}} A'$, there exists a trace $B \xRightarrow{\text{tr}} B'$ such that $A' \sim B'$. We say that A and B are *trace equivalent*, written $A \approx_t B$, when $A \sqsubseteq_t B$ and $B \sqsubseteq_t A$.

DEFINITION 2.10 (Simulation, (Bi)similarity). A *labelled simulation* (or simply *simulation*) is a relation \mathcal{R} such that for all extended processes A, B , $A \mathcal{R} B$ entails

1. $A \sim B$
2. for all transitions $A \xrightarrow{\alpha} A'$, there exists a trace $B \xRightarrow{\alpha} B'$ such that $A' \mathcal{R} B'$

We call \sqsubseteq_s (*simulation preorder*) the largest simulation, and \approx_s (*labelled similarity*, or simply *similarity*) the relation $\sqsubseteq_s \cap \sqsupseteq_s$. *Bisimilarity* \approx_b is the largest symmetric simulation.

Note in particular that

$$\approx_b \subset \approx_s \subset \approx_t$$

i.e. two bisimilar processes are always similar, and two similar processes are always trace equivalent. These equivalences are well established as means to express security properties [4, 1]. Trace equivalence has been studied intensively for security protocols [28, 7, 29, 31] while, for example, labelled bisimilarity is used as a characterisation for *observational equivalence* [1].

Each equivalence implies slightly different adversaries. As shown in [29], \approx_t characterizes may-testing, i.e., equivalence in the presence of an arbitrary adversarial process running in parallel. \approx_b characterizes observational equivalence [1] and considers a more adaptive adversary; \approx_b was also introduced as a proof technique for may-testing in [4]. Finally, it was recently shown [30] that \approx_s characterizes a may-testing equivalence in the presence of a probabilistic adversary, i.e. an adversarial process that is allowed to branch probabilistically.

EXAMPLE 2.11. We refer again to the processes modelling the Private Authentication protocol as described in Example 2.3. We let for instance the processes $P_a = B(sk_B, pk(sk_A), N_B, r_B)$ and $P_c = B(sk_B, pk(sk_C), N_B, r_B)$ modelling the role of B accepting connections from A and C , respectively. We want to verify whether an adversary would be able to distinguish the two situations. This could be modelled for example by

$$(\{P_a\}, \Phi_0) \approx_t (\{P_c\}, \Phi_0) \quad \text{with } \Phi_0 = \{ax_1 \mapsto pk(sk_A), ax_2 \mapsto pk(sk_B), ax_3 \mapsto pk(sk_C)\}$$

The initial frame Φ_0 models that the attacker knows the public keys of all agents. It appears that this equivalence statement holds, the core argument being that for all messages u_1, u_2, r_1, r_2 and $pk_1, pk_2 \in \{pk(sk_A), pk(sk_B), pk(sk_C)\}$, the following frames are statically equivalent:

$$\Phi_0 \cup \{ax_4 \mapsto \text{aenc}(u_1, r_1, pk_1)\} \quad \Phi_0 \cup \{ax_4 \mapsto \text{aenc}(u_2, r_2, pk_2)\}$$

In particular this equivalence statement still holds if we weaken the cryptographic assumptions on aenc by assuming that a ciphertext is distinguishable from an arbitrary term, which is modelled by adding the rewrite rule $\text{test_aenc}(\text{aenc}(x, y, pk(z))) \rightarrow \text{ok}$. However trace equivalence is violated if we add the rule $\text{get_key}(\text{aenc}(x, y, pk(z))) \rightarrow pk(z)$. A possible attack trace is, with $N, R \in \mathcal{F}_0$:

$$\begin{array}{l} (\{P_a\}, \Phi_0) \xrightarrow{c(\text{aenc}(\langle N, ax_1 \rangle, R, ax_2))} (\{\bar{c}\langle u \rangle\}, \Phi_0) \quad \text{with } u = \text{aenc}(\langle N, N_B, pk(sk_B) \rangle, r_B, pk(sk_A)) \\ \xrightarrow{\bar{c}\langle ax_4 \rangle} (\{\mathbf{0}\}, \Phi) \quad \text{with } \Phi = \Phi_0 \cup \{ax_4 \mapsto u\} \end{array}$$

Indeed there is only one trace in the other process taking the same actions:

$$\begin{aligned} (\{P_c\}, \Phi_0) &\xrightarrow{c(\text{aenc}(\langle N, ax_1 \rangle, R, ax_2))} (\{\bar{c}\langle v \rangle\}, \Phi_0) && \text{with } v = \text{aenc}(N_B, r_B, \text{pk}(sk_B)) \\ &\xrightarrow{\bar{c}\langle ax_4 \rangle} (\{0\}, \Psi) && \text{with } \Psi = \Phi_0 \cup \{ax_4 \mapsto v\} \end{aligned}$$

and $\Phi \neq \Psi$ because the recipe $\xi = \text{get_key}(ax_4)$ is evaluated to $\text{pk}(sk_A)$ in Φ and to $\text{pk}(sk_B)$ in Ψ . That is, the recipes ξ and $\zeta = ax_4$ are equal in Φ but not in Ψ . \blacklozenge

In practice: security goals for Private Authentication We now demonstrate in more details how equivalence properties can be used to model security in practical scenarios through a complete case study. We model the three security goals of the Private Authentication Protocol described in Example 2.3. For simplicity we present the simplest scenario of a single session of the protocol in this section (i.e. only one instance of the roles of A and B communicating in parallel). Of course a more extensive analysis needs to consider more parallel sessions. In the following we write pk_X and sk_X the public and private keys of an identity X and

$$P(A, C, N_A, r_A, B, D, N_B, r_B) = X(sk_A, pk_C, N_A, r_A) \mid B(sk_B, pk_D, N_B, r_B)$$

the process that runs in parallel the roles of A attempting to initiate a communication with C and B accepting a connection from a unique identity D . We assume an initial frame Φ_0 that contains the public keys of all identities involved in the process.

The security goals state that the protocol should conceal the identities of the participants (including C the recipient of A and D the connection accepted by B) and the values of the exchanged nonces. A possible formalisation is that there should not be any observable difference in $P(A, C, N_A, r_A, B, D, N_B, r_B)$ when replacing the identities by others and N_A, N_B by any other value. That is, for all identities $A, B, C, D, A', B', C', D'$, all terms N_A, N_B, N'_A, N'_B , and fresh names r_A, r_B ,

$$(\{P(A, C, N_A, r_A, B, D, N_B, r_B)\}, \Phi_0) \approx (\{P(A', C', N'_A, r_A, B', D', N'_B, r_B)\}, \Phi_0)$$

where \approx is either \approx_t , \approx_s or \approx_b and Φ_0 is a frame whose image contains the public keys of all identities involved. This models a form of non-interference property and has been called *strong secrecy* in [18].

2.4 Complexity and decision problems

So far we detailed how process equivalences can be used to model privacy preservation in security protocols. Our goal in this article is to present decidability and complexity results for static equivalence, trace equivalence and labelled bisimilarity.

On sizes Before going further we need to clarify the notion of *size* of the inputs since it plays a central role in complexity analyses. This is particularly important for our purpose since there

exist several conventions for representing terms. The *tree size* of term t refers to its number of symbols and is written $|t|$. It corresponds to a classical representation of a term as a tree. On the other hand some of our complexity results are stated w.r.t. a succinct representation of terms as Directed Acyclic Graphs (DAG) with maximal sharing (which may be exponentially more concise). If $st(t)$ is the set of subterms of t , the *DAG size* of t refers to the cardinality $|st(t)|$ and is written $|t|_{\text{dag}}$. This definition is lifted to sets and sequences of terms with the sharing common to all elements of the structure. The size of a signature \mathcal{F} is the sum of the arities of the symbols of \mathcal{F} (which is finite since \mathcal{F}_c and \mathcal{F}_d are finite) and the size of a rewrite system \mathcal{R} is the sum of the sizes of the two hand sides of its rules. The size of a process is the sum of the number of operators of the process and of the sizes of all terms appearing in the process (in conditionals, channels, and output terms). We emphasise that

- A complexity upper bound stated w.r.t. the DAG size of the inputs is a stronger result than the same upper bound stated w.r.t. the tree size.
- On the contrary a complexity lower bound stated in DAG size is a weaker result than the corresponding result in tree size.

In this article we only address the strongest configurations: lower bounds in the tree representation of terms, upper bounds in DAG.

Complexity classes We now shortly remind some background about complexity, mainly introducing our notations. Given $f : \mathbb{N} \rightarrow \mathbb{N}$, we define $\text{TIME}(f)$ (resp. $\text{SPACE}(f)$) the class of problems decidable by a deterministic Turing machine running in time (resp. in space) at most $f(n)$ where n is the size of the parameters of the problem. It is common to define the following classes:

$$\begin{aligned} \text{LOGSPACE} &= \bigcup_{p \in \mathbb{N}} \text{SPACE}(\log(n^p)) & \text{PTIME} &= \bigcup_{p \in \mathbb{N}} \text{TIME}(n^p) \\ \text{PSPACE} &= \bigcup_{p \in \mathbb{N}} \text{SPACE}(n^p) & \text{EXPTIME} &= \bigcup_{p \in \mathbb{N}} \text{TIME}(2^{n^p}) \end{aligned}$$

One can define their non-deterministic counterparts NLOGSPACE (NL for short), NPTIME , NPSPACE and NEXPTIME . Given a (non-deterministic) class C , we call $\text{co-}C$ the class of problems whose negation is in C . From now on we often omit the suffix TIME in the name of time complexity classes for the sake of succinctness. Then it is known that:

$$\text{LOGSPACE} \subseteq \text{NL} = \text{coNL} \subseteq \text{P} \subseteq \text{NP,coNP} \subseteq \text{PSPACE} = \text{NPSPACE} \subseteq \text{EXP} \subseteq \text{NEXP,coNEXP}$$

To define complete problems for complexity classes above PTIME we use classical *many-to-one polytime reductions*. We also mention the notion of *oracle* reduction, deciding a problem with a constant-time black box for another problem: the class of problems decidable in C with an oracle for a problem Q is noted C^Q . When Q is complete for a class \mathcal{D} w.r.t. a notion of reduction executable in C , we may write $C^{\mathcal{D}}$ instead; in particular $C^{\mathcal{D}} = C^{\text{co}\mathcal{D}}$. This kind of reduction is

needed to define the last complexity classes we will use in this article: the *polynomial hierarchy*, which is a collection of complexity classes between PTIME and PSPACE. Indeed the difference between NP and PSPACE lies in their capacity to express quantifier alternation; the usual complete problems considered for these two complexity classes are, given a boolean formula φ :

- SAT (NP complete): does $\exists x_1, \dots, x_n. \varphi(x_1, \dots, x_n)$ hold?
- QBF (PSPACE complete): does $\forall x_1, \exists y_1, \dots, \forall x_n, \exists y_n. \varphi(x_1, y_1, \dots, x_n, y_n)$ hold?

The polynomial hierarchy characterises all classes corresponding to intermediate alternations.

DEFINITION 2.12. The polynomial hierarchy PH consists of the classes Σ_n defined by $\Sigma_0 = \text{PTIME}$ and $\Sigma_{i+1} = \text{NP}^{\Sigma_i}$. In particular, $\Sigma_1 = \text{NP}$. We also write Π_i for $\text{co}\Sigma_i$.

Problems studied in this article We thus study the following decision problems:

STATEQ:

- ▷ INPUT: A rewriting system \mathcal{R} , two frames Φ and Ψ .
- ▷ QUESTION: $\Phi \sim \Psi$ for \mathcal{R} ?

TRACEQ:

- ▷ INPUT: A rewriting system \mathcal{R} , two processes P and Q .
- ▷ QUESTION: $(\{\{P\}\}, \emptyset) \approx_t (\{\{Q\}\}, \emptyset)$ for \mathcal{R} ?

We also consider TRACEINCL, SIMULATION, SIMILARITY, BISIMILARITY to be the analogue problems of TRACEQ, replacing trace equivalence by the relations \sqsubseteq_t , \sqsubseteq_s , \approx_s , and \approx_b , respectively. As we explained previously these problems are undecidable in general and we need to put restrictions on the inputs, in addition to the restriction to a bounded number of sessions, which is inherent to our model. Typically our results all include the restriction (inherent to our model) to constructor destructor theories and bounded processes. When we say for example that “TRACEQ is decidable for constructor destructor subterm convergent rewriting systems” it means that we are studying the following decision problem:

- ▷ INPUT: A constructor destructor subterm convergent rewriting system \mathcal{R} , two processes P and Q .
- ▷ QUESTION: Are P and Q trace equivalent (for \mathcal{R})?

The way we state the problem implies that complexity analyses need to account for the size of all inputs, including the rewriting system. However the treatment of this question is not uniform in the literature. Complexity analyses in [2, 15, 29] consider the rewriting system as a constant of the problem. For the example above this means considering, for each constructor destructor subterm convergent rewriting system \mathcal{R} , the following decision problem:

- ▷ INPUT: Two bounded processes P and Q .
- ▷ QUESTION: Are P and Q trace equivalent (for \mathcal{R})?

For this formulation of the problem, generic completeness results w.r.t. complexity classes are not possible in general because different complexities may arise for each rewriting system \mathcal{R} . This is for example the case in [2], where `STATEq` is proven PTIME for any fixed subterm convergent rewriting system: the problem is indeed PTIME-hard for some of them [34] but also LOGSPACE for others as we prove it in this article. All existing procedures [2, 39, 42] are actually exponential in the size of the rewriting system. This is why we refer to this problem as *parametric equivalence* and say by opposition that *general equivalence* is the initial variant with the rewriting system considered as part of the input. We argue that the latter is more relevant today as the rewriting system can now be specified by the user in many automated tools. This motivated for example to prove in [34] that the complexity results of [15, 29] (stated in the parametric setting) were also valid in the general setting.

3. Structure of the decision procedure

We detail in this section our overall decision procedure for equivalence properties, intuitively reducing them to solving some forms of symbolic constraints. We express this through a novel notion of *partition tree* that crisply characterises equivalence proofs. We formalise in this section the main properties of this tree and describe how to derive an actual decision procedure from it; the constraint solving procedure necessary to generate the tree itself is then later detailed in Section 4.

3.1 The symbolic approach for decidability

Our decision procedures rely on a *symbolic semantics*, by opposition to the usual semantics of the calculus (recall Figure 1) that we will call the *concrete semantics* from now on. Specifically, rather than fetching concrete input terms from the active attacker, our symbolic semantics abstract these inputs and only record the constraints they should satisfy to execute the protocol. This thus provides a finite representation of the infinite set of actions potentially available to the attacker. For example let $c \in \mathcal{F}_0$, $h/1 \in \mathcal{F}_c$, $k \in \mathcal{N}$ and consider the process

$$P = \bar{c}\langle k \rangle. c(x). \text{if } \text{fst}(x) = k \text{ then } \bar{c}\langle h(x) \rangle$$

The trace executing the output $h(x)$ will gather constraints that intuitively indicate that: 1. x is a term deducible by the attacker from the frame $\{ax_1 \mapsto k\}$; and 2. $x = \langle k, y \rangle$ for some term y . A constraint solving algorithm, detailed in Section 4.2, can then be used to show that these constraints have a *solution*: the recipe $\xi = \langle ax_1, a \rangle$, $a \in \mathcal{F}_0$, can be used to compute the input term x and satisfy the constraints, which justifies that the output of $h(x)$ is reachable. Similar approaches are common to decide reachability or equivalence properties of bounded

processes [15, 29]; our approach is however more widely applicable due to our absence of syntactic restrictions on processes.

Formalising symbolic constraints We first introduce a new type of variables, used in recipes:

DEFINITION 3.1 (second-order terms). We consider a partition of the set of (non-axiom) variables $\mathcal{X} \setminus \mathcal{AX} = \mathcal{X}^1 \uplus \mathcal{X}^2$. The elements of \mathcal{X}^1 are called *first-order variables* and correspond to those we used so far in terms (in processes, frames, rewrite rules). Those of \mathcal{X}^2 are called *second-order variables* and are used to represent an undefined recipe. A *first-order term* is an element of $\mathcal{T}(\mathcal{F} \cup \mathcal{F}_0 \cup \mathcal{N} \cup \mathcal{X}^1)$ and a *second-order term* is an element of $\mathcal{T}(\mathcal{F} \cup \mathcal{F}_0 \cup \mathcal{AX} \cup \mathcal{X}^2)$.

We now distinguish $\text{vars}^1(u) = \text{vars}(u) \cap \mathcal{X}^1$, $\text{vars}^2(u) = \text{vars}(u) \cap \mathcal{X}^2$, and $\text{axioms}(u) = \text{vars}(u) \cap \mathcal{AX}$. Note that we say that a second-order term t is ground if $\text{vars}^2(t) = \emptyset$, i.e., t may contain axioms. By definition, a recipe is therefore a ground second-order term. We also adapt the other notations of the term algebra to reflect the separation: st^1, st^2, \dots

In practice, when executing an input instruction $c(x)$ in the symbolic semantics, x will be associated to a fresh second-order variable written $X:i$, where $X \in \mathcal{X}^2$ will serve as a placeholder for the recipe used to compute x , and $i \in \mathbb{N}$ indicates that only the first i axioms of the frame are available to compute the recipe in question. This is formalised by the following, natural extension of the notion of substitution:

DEFINITION 3.2 (second-order substitutions). We suppose a partition $\mathcal{X}^2 = \uplus_{i \in \mathbb{N}} \mathcal{X}_{=i}^2$ where each class $\mathcal{X}_{=i}^2$ is infinite. We also write $\mathcal{X}_{\leq i}^2 = \bigcup_{j=0}^i \mathcal{X}_{=j}^2$. If X is a second-order variable we may write $X:i$ to emphasise that $X \in \mathcal{X}_{=i}^2$ and say in this case that X is of type i . A *second-order substitution* is then a substitution Σ of domain $\text{dom}(\Sigma) \subseteq \mathcal{X}^2$ that *respects types*:

$$\forall X:i \in \text{dom}(\Sigma), X\Sigma \in \mathcal{T}_i^2 \quad \text{where } \mathcal{T}_i^2 = \mathcal{T}(\mathcal{F} \cup \mathcal{F}_0 \cup \mathcal{X}_{\leq i}^2 \cup \{\text{ax}_1, \dots, \text{ax}_i\})$$

Altogether, we can then define the constraints that we use to characterise the possible values that an input term x may take:

DEFINITION 3.3 (atoms). We consider the following three kinds of atoms:

1. *deduction fact* $\xi \vdash^? u$ where u is a message in normal form and ξ is a second-order term such that $\text{root}(\xi) \notin \mathcal{F}_c$;
2. *second-order equations* $\xi =^? \zeta$ where ξ and ζ are two second-order terms;
3. *(first-order) equations* $u =^? v$ where u and v are two first-order terms (not necessarily messages).

The negation $\neg(\alpha =^? \beta)$ of an equation is written $\alpha \neq^? \beta$ and called a *disequation*.

DEFINITION 3.4 (constraint). An *atomic constraint* (or an *atomic formula*) is an atom that is either a deduction fact, a second-order equation, or a first-order equation $u =^? v$ where u and v

are constructor terms. A *constraint* is then a first-order formula over atomic constraints, that is, either an atomic constraint, \top , \perp , or of the form $\varphi \wedge \psi$, $\varphi \vee \psi$, $\neg\varphi$, $\forall x.\varphi$, or $\forall X:n.\varphi$ for φ, ψ constraints. Note that $\text{vars}(\varphi)$ then refers to the *free* variables of the constraint φ .

A deduction fact $\xi \vdash^? u$ indicates that term u is deducible by the recipe ξ and second-order equations $\xi =^? \zeta$ are used to put restrictions on which recipes ξ may be used to do so. For example $X:i \vdash^? x$ states that the variable x is to be replaced by a term deducible by the attacker using at most the i first outputs of the frame; a constraint solving procedure may then impose that $\exists Y:i. X =^? f(Y)$, i.e., that the underlying recipe should have a f symbol at its root. Equations reflect the syntactic equalities that the first-order terms verify. Typically when executing $\text{fst}(x) = t$ then P else Q , the positive branch will intuitively lead to the constraint $\exists y. x =^? \langle t, y \rangle$ and the negative branch to $\forall y. x \neq^? \langle t, y \rangle$.

Constraint systems Finally we define and give some properties of *constraint systems* that are used to collect the first-order constraints induced by a given execution of a process.

DEFINITION 3.5 (constraint system). A *constraint system* is a triple $C = (\Phi, D, E^1)$ whose elements are of the following form:

1. $\Phi = \{ax_1 \mapsto t_1, \dots, ax_n \mapsto t_n\}$ is a frame (not necessarily ground)
2. D is a set of constraints of the form $X \vdash^? x$, with $X \in \mathcal{X}_{\leq n}^2$, $x \in \mathcal{X}^1$. We also require the *origination property*: for all $i \in \llbracket 1, n \rrbracket$, for all $x \in \text{vars}(t_i)$, there exists $X \in \mathcal{X}_{\leq i-1}^2$ such that $(X \vdash^? x) \in D$.
3. E^1 is a set of constraints of the form $u =^? v$ or $\forall z_1 \dots \forall z_k. \bigvee_{j=1}^r u_j \neq^? v_j$.

The components of C are also written $\Phi(C)$, $D(C)$ and $E^1(C)$. The set D contains all input binders x that have been executed, each mapped to a second-order variable X that will serve as a placeholder for the corresponding recipe. Next the origination property expresses that when reference is made to an input x in an output t_i , this input should be computed only from the previous outputs t_1, \dots, t_{i-1} . This is a natural invariant preventing cyclic input-output dependencies, always satisfied in practice. Finally E^1 is a set of (dis)equalities imposed on the protocol messages by conditionals, among others. We will formalise in Section 3.3 the semantics of these constraints through a notion of *solution*.

REMARK 3.6 (notational conventions). We use several convenient notations throughout the article to lighten the presentation of constraints. First of all we do not make a difference between sets and conjunctions of constraints: for instance we may write $E^1 = \bigwedge_{i=1}^n \varphi_i$ instead of $E^1 = \{\varphi_i\}_{i=1}^n$ and conversely. We also interpret a substitution σ as the set of equations $\mathcal{E} = \{x =^? x\sigma \mid x \in \text{dom}(\sigma)\}$.

3.2 (Most general) unifiers

We now recall some basics on term unification, a key concept in symbolic models that has some specificities in our context, in particular regarding second-order terms.

Unification of first-order terms Two first-order terms u and v are *unifiable* if there exists a substitution σ , called a *unifier*, such that $u\sigma = v\sigma$. For example the terms $u = \langle \text{sdec}(x, y), z \rangle$ and $v = \langle z_1, z_2 \rangle$ are unified by $\sigma = \{z_1 \mapsto \text{sdec}(x, y), z_2 \mapsto z\}$. The terms u and z' are unifiable as well using $\sigma = \{z' \mapsto u\}$, but the terms u and z are not. More generally, a unifier σ of a set of equations $\mathcal{E} = \{u_i =^? v_i\}_{i=1}^n$ is a unifier of u_i and v_i for all i . A classical characterisation of the set of unifiers of two terms is based on *most general unifiers*:

DEFINITION 3.7 (most general unifier). A unifier σ of \mathcal{E} is said to be a *most general* one if, for any θ unifier of \mathcal{E} , there exists τ such that $\theta = \sigma\tau$. In this case, we write $\sigma = \text{mgu}(\mathcal{E})$ (and it is unique up to variable renaming). When \mathcal{E} is not unifiable, we write $\text{mgu}(\mathcal{E}) = \perp$.

A straightforward inductive procedure allows to decide whether \mathcal{E} is unifiable and, if it is, to compute $\text{mgu}(\mathcal{E})$. We assume w.l.o.g. that this computation does not introduce variables, that is, if $\sigma = \text{mgu}(\mathcal{E})$ then $\text{dom}(\sigma) \cup \text{vars}(\text{img}(\sigma)) \subseteq \text{vars}(\mathcal{E})$. We also require that $\text{dom}(\sigma) \cap \text{vars}(\text{img}(\sigma)) = \emptyset$, that is, applying a mgu twice has no more effect than applying it once. Note as well that all unifiers are instances of the mgu but the converse is also true, that is, all instances of a mgu are unifiers. By convenience we also write $\text{mgu}(\mathcal{E})$ in the case where \mathcal{E} contains disequations (typically when writing $\text{mgu}(\mathbb{E}^1(C))$): in this case only equations are taken into account and nothing ensures that the mgu satisfies the disequations of \mathcal{E} .

However mgus are only syntactic: when taking the rewriting system \mathcal{R} into account we say that σ is a *unifier modulo theory* of \mathcal{E} when for all $(u =^? v) \in \mathcal{E}$, $u\sigma \downarrow = v\sigma \downarrow$. A standard procedure based on narrowing (not detailed here) allows to compute *most general unifiers modulo \mathcal{R}* when \mathcal{R} is subterm convergent among others [41]. However unlike the syntactic case they are not unique in general:

DEFINITION 3.8 (most general unifier modulo theory). We let \mathcal{E} be a set of equations and \mathcal{R} be a convergent rewriting system. A set of *most general unifiers modulo \mathcal{R}* is a set of substitutions $\text{mgu}_{\mathcal{R}}(\mathcal{E})$ that verifies the following properties:

1. for all $\sigma \in \text{mgu}_{\mathcal{R}}(\mathcal{E})$, σ is a unifier of \mathcal{E} modulo \mathcal{R}
2. for all θ unifier of \mathcal{E} modulo \mathcal{R} , there exists $\sigma \in \text{mgu}_{\mathcal{R}}(\mathcal{E})$ and a substitution τ such that for all $x \in \text{vars}(\mathcal{E})$, $x\theta \downarrow = x\sigma\tau \downarrow$

Again we emphasise that equality modulo \mathcal{R} only operates on valid messages, that is, if $\sigma \in \text{mgu}_{\mathcal{R}}(u =^? v)$ then $u\sigma$ and $v\sigma$ verify the msg predicate. A typical use case we consider in the symbolic semantics is $\text{mgu}_{\mathcal{R}}(u =^? u)$, which is the most general substitution σ such that

$\text{msg}(u\sigma)$ holds (if any). For example if $u = \text{adec}(x, y)$ we have $\text{mgu}_{\mathcal{R}}(u =? u) = \{\sigma\}$, where:

$$\sigma = \{x \mapsto \text{aenc}(x', x_r, \text{pk}(y')), y \mapsto y'\} \quad x', x_r, y' \in \mathcal{X} \text{ fresh}$$

This example also highlights that, unlike the syntactic case, computing mgu's modulo theory may require to introduce new variables. This also makes it possible to enforce that $\text{dom}(\sigma) \cap \text{vars}(\text{img}(\sigma)) = \emptyset$.

Unification of second-order terms Intuitively, the unification of two second-order terms ξ and ζ modulo theory means that they deduce the same first-order term u w.r.t. a given frame Φ . This unusual kind of unification is performed as a part of our constraint solving algorithm using a dedicated kind of constraint written $\xi =?_f \zeta$, detailed in Section 4.1.1. However, even the computation of syntactic mgu's has some subtleties for second-order terms that we discuss below.

As in the first-order case, a syntactic unifier of ξ and ζ is a second-order substitution Σ such that $\xi\Sigma = \zeta\Sigma$. However, computing Σ is not as simple as usual due to the variable types. Indeed, we recall that by definition, a second-order substitution has to respect types, that is, a variable $X:n$ cannot be mapped to a term containing axioms ax_i or variables $Y:i$ if $i > n$. Say for instance we want to unify the two second-order terms $X:1$ and $f(Y:2)$: a regular computation of the mgu would yield the substitution $\Sigma = \{X \mapsto f(Y)\}$, which does not respect the type of X . In this case, one solution is to introduce a fresh variable $Z:1$ and to choose the following unifier:

$$\text{mgu}(X =? f(Y)) = \{X \mapsto f(Z), Y \mapsto Z\} = \Sigma\{Y \mapsto Z\}.$$

Given a second-order term ξ , let us write $\#(\xi)$ the maximal type of second-order variables and axioms appearing in ξ , that is, the minimal type i such that $\xi \in \mathcal{T}_i^2$. Formally,

$$\#(\xi) = \min\{i \in \mathbb{N} \mid \xi \in \mathcal{T}_i^2\}.$$

The mgu of a conjunction of equations φ is then computed inductively as follows:

$$\text{mgu}(\top) = \top$$

$$\text{mgu}(\varphi \wedge f(\xi_1, \dots, \xi_n) =? g(\zeta_1, \dots, \zeta_n)) = \begin{cases} \perp & \text{if } f \neq g \\ \text{mgu}(\varphi \wedge \bigwedge_{i=1}^n \xi_i =? \zeta_i) & \text{if } f = g \end{cases}$$

$$\text{mgu}(\varphi \wedge X:i =? \xi) = \begin{cases} \perp & \text{if } X \in \text{vars}^2(\xi) \text{ and } \xi \neq X \\ \perp & \text{else if } \exists \text{ax}_j \in \text{axioms}(\xi), j > i \\ \Sigma_0 \Sigma & \text{else if } \xi \notin \mathcal{X}^2, Y:j \in \text{vars}^2(\xi), j > i, Z:i \text{ fresh and} \\ & \text{with } \Sigma_0 = \{Y \mapsto Z\} \text{ and } \Sigma = \text{mgu}(\varphi \Sigma_0 \wedge X:i =? \xi \Sigma_0) \\ \Sigma_0 \Sigma & \text{else if } \#(\xi) \leq i, \text{ with } \Sigma_0 = \{X \mapsto \xi\} \text{ and } \Sigma = \text{mgu}(\varphi \Sigma_0) \end{cases}$$

As before we extend this notation to arbitrary sets \mathcal{E} , that is, we may write $mgu(\mathcal{E})$ even if \mathcal{E} contains disequations (which are then ignored during the computation). The correctness of this function is proved below.

PROPOSITION 3.9 (correctness of second-order mgu's). *For all sets of second-order equations \mathcal{E} , the computation of $mgu(\mathcal{E})$ terminates. Besides we have that $mgu(\mathcal{E}) = \perp$ iff there exist no unifiers of \mathcal{E} . When $mgu(\mathcal{E}) \neq \perp$, we have that:*

1. $mgu(\mathcal{E})$ is a second-order substitution, i.e., it respects types, and it is a unifier of \mathcal{E} ;
2. for all unifiers Σ of \mathcal{E} , there exists a second-order substitution Σ_0 such that $\Sigma = mgu(\mathcal{E})\Sigma_0$.

PROOF. We only prove the termination since all other properties can be proved separately by straightforward inductions on the definition of mgu . We let the partial ordering on second order variables \leq given by the types, i.e. $X:i \leq Y:j$ iff $i \leq j$. Given a set of second-order equations \mathcal{E} we then let

$$\mu(\mathcal{E}) = (\text{vars}^2(\mathcal{E}), M(\mathcal{E}), F(\mathcal{E}))$$

where $M(\mathcal{E})$ is the multiset of variables of \mathcal{E} , i.e. multiplicity included, and $F(\mathcal{E})$ is the multiset of the sizes of the equations of \mathcal{E} (where the size of $\xi =^? \zeta$ is the number of function symbols in ξ and ζ). The first two components are ordered w.r.t. the multiset extension of \leq , and the third one w.r.t. the multiset extension of \leq . The overall tuple is ordered w.r.t. the lexicographic composition of the three components.

If we number from 1 to 7 the axioms defining mgu , we can show that μ decreases at each recursive call: (1), (2), (4) and (5) make no recursive calls; (3) preserves vars^2 and M , and makes F decrease; (6) replaces all occurrences of Y with Z that has a lower type which makes vars^2 decrease. Regarding (7) two cases can arise: either $\xi = X$ or $X \notin \text{vars}^2(\xi)$. In the first case vars^2 is non increasing and M is decreasing since two occurrences of X are removed and the rest of the formula \mathcal{E} is left unchanged. In the second case vars^2 is decreasing since all occurrences of X are removed and no variables are added. ■

3.3 (Most general) solutions

Solutions Let us now formalise the semantics of constraints. Given a constraint φ , a frame Φ and second- and first-order substitutions Σ and σ we define the predicate $(\Phi, \Sigma, \sigma) \models \varphi$ by:

$$\begin{aligned} (\Phi, \Sigma, \sigma) \models \xi \vdash^? u & \quad \text{iff} \quad \xi \Sigma \Phi \sigma \downarrow = u \sigma \downarrow \\ (\Phi, \Sigma, \sigma) \models \xi =^? \zeta & \quad \text{iff} \quad \xi \Sigma = \zeta \Sigma \\ (\Phi, \Sigma, \sigma) \models u =^? v & \quad \text{iff} \quad u \sigma = v \sigma \\ (\Phi, \Sigma, \sigma) \models \forall x. \varphi & \quad \text{iff} \quad \text{for all first-order ground terms } t, (\Phi, \Sigma, \sigma) \models \varphi \{x \mapsto t\} \\ (\Phi, \Sigma, \sigma) \models \forall X:n. \varphi & \quad \text{iff} \quad \text{for all } \xi \in \mathcal{T}_n^2, (\Phi, \Sigma, \sigma) \models \varphi \{X \mapsto \xi\} \end{aligned}$$

The definition is extended with logical connectives $\neg, \wedge, \vee, \dots$ in the natural way. By convention, writing $(\Phi, \Sigma, \sigma) \models \varphi$ implicitly assumes that, for all $x \in \text{vars}^1(\varphi)$ and $X \in \text{vars}^2(\varphi)$, $x\sigma$ and

$X\Sigma\Phi\sigma$ are ground. Intuitively the second-order substitution Σ describes which recipes are used to deduce each input term appearing in φ , while σ gives the actual values of these inputs.

DEFINITION 3.10 (solution of a constraint system). We say that (Σ, σ) is a *solution* of C if $\text{dom}(\Sigma) = \text{vars}^2(C)$, $\text{dom}(\sigma) = \text{vars}^1(C)$ and $(\Phi(C), \Sigma, \sigma) \models D(C) \wedge E^1(C)$. We call Σ a *second-order solution* of C and σ its *first-order solution*. The set of solutions of C is written $\text{Sol}(C)$.

The solutions of a constraint system C indicate how the inputs of C (i.e., $\text{vars}^1(D(C))$) can be computed while satisfying the constraints imposed by $E^1(C)$. Due to the origination property, the values the first-order solution σ takes on $\text{vars}^1(D(C))$ is uniquely determined by which recipes are used to deduce terms, i.e., by the second-order solution Σ .

EXAMPLE 3.11. Consider again the example $P = \bar{c}\langle k \rangle. c(x). \text{if fst}(x) = k \text{ then } \bar{c}\langle h(x) \rangle$. The traces performing the final output $h(x)$ are characterised by the constraint system

$$\Phi(C) = \{ax_1 \mapsto k, ax_2 \mapsto h(x)\} \quad D(C) = \{X \vdash^? x\} \quad E^1(C) = \{x =^? \langle k, y \rangle\}$$

where $X:1$ and y are fresh second- and first-order variables, respectively. Observe in particular that the informal constraint “*there exists a term y such that $x = \langle k, y \rangle$ ” is not formalised using an explicit \exists quantification but with a free variable y . All second-order solutions of C are instances of $\Sigma_0 = \{X \mapsto \langle ax_1, Y \rangle\}$ where $Y:1$ is fresh, for example, $\Sigma = \{X \mapsto \langle ax_1, a \rangle\}$ with $a \in \mathcal{F}_0$. The corresponding first-order solution is then $\sigma = \{x \mapsto \langle k, a \rangle, y \mapsto a\}$. \blacklozenge*

Most general solutions Similarly to mgu’s, we now introduce a novel characterisation of solutions as instances of so-called most general solutions (*mgs*). The definition is parametrised with a predicate π on second-order substitutions, writing $\text{Sol}^\pi(C) = \{(\Sigma, \sigma) \in \text{Sol}(C) \mid \pi(C) \text{ holds}\}$. Filtering solutions this way will essentially permit, during the decision procedure, to perform case analyses on the form of the solutions.

DEFINITION 3.12 (most general solution). A set of *most general solutions* of C that satisfy π is a set $\text{mgs}^\pi(C)$ of second-order substitutions such that:

1. for all $\Sigma_0 \in \text{mgs}^\pi(C)$, $\text{dom}(\Sigma_0) \subseteq \text{vars}^2(C)$, for all injections Σ_1 to fresh constants and of domain $\text{dom}(\Sigma_1) = \text{vars}^2(\text{img}(\Sigma_0), C) \setminus \text{dom}(\Sigma_0)$, $(\Sigma_0\Sigma_1, \sigma) \in \text{Sol}^\pi(C)$ for some σ .
2. for all $(\Sigma, \sigma) \in \text{Sol}^\pi(C)$, there exists $\Sigma_0 \in \text{mgs}^\pi(C)$ and Σ_1 such that $\Sigma = \Sigma_0\Sigma_1$.

We omit the predicate π in the case where $\pi = \top$, i.e., $\pi(\Sigma)$ holds for any substitution.

The first condition of the definition states that a mgs Σ_0 is “almost” a solution of C : Σ_0 is allowed to be given in a minimal form that does not instantiate all variables of $\text{vars}^2(C)$, and that may not have a ground image; but we obtain a solution by replacing all pending variables by fresh names using Σ_1 . The second condition states that all solutions are instances of a mgs.

EXAMPLE 3.13. In Example 3.11 we have $mgs(C) = \{\Sigma_0\}$ and $Sol(C)$ is the set of all ground instances of Σ_0 . However in general the situation may be less ideal. For example a constraint system may have several most general solutions; a simple example being, with $h/1$ and $k \in \mathcal{N}$:

$$\Phi(C) = \{ax_1 \mapsto h(k), ax_2 \mapsto k\} \quad D(C) = \{X:2 \vdash^? x\} \quad E^1(C) = \{x =^? h(k)\}$$

The constraint system C expresses that an input x should be instantiated by $h(k)$, potentially by using the two previous outputs $h(k)$ and k . There are therefore two ways of computing x : either using ax_1 or $h(ax_2)$, which is reflected as the fact that $mgs(C) = \{\Sigma_1, \Sigma_2\}$ with

$$\Sigma_1 = \{X \mapsto h(ax_2)\} \quad \Sigma_2 = \{X \mapsto ax_1\}$$

Still, it is possible to obtain unique mgs' by performing a case analysis and restricting the solutions accordingly; typically here we have $mgs^{\pi_i}(C) = \{\Sigma_i\}$ with

$$\pi_1(\Sigma) \triangleq \exists X'. X =^? h(X') \quad \pi_2(\Sigma) \triangleq \forall X'. X \neq^? h(X')$$

Another notable point is that some ground instances of a mgs may not be solutions themselves. Taking $a \in \mathcal{F}_0$ a simple example is given by $C = (\emptyset, X \vdash^? x, x \neq^? a)$ and $mgs(C) = \{id\}$: the substitution $\{X \mapsto a\}$ is a ground instance of the identity but not a solution (which does not contradict Item 1 of Definition 3.12 since although a is a constant, it is not fresh). \blacklozenge

We describe in Section 4.2 how to generate a finite set of most general solutions, at least in the context of our decision procedure.

3.4 Symbolic semantics

Symbolic execution We now describe formally our symbolic semantics. It shares some common ground with the concrete semantics of the calculus, except that a constraint system collects the execution's constraints. The semantics operates on so-called *symbolic processes* (\mathcal{P}, C) where \mathcal{P} is a multiset of (non-necessarily ground) plain processes and C is a constraint system. All free variables of \mathcal{P} are bound by deductions facts, that is, for all $x \in vars(\mathcal{P})$ there exists $(X \vdash^? x) \in D(C)$. The semantics then takes the form of a labelled transition system $\xrightarrow{\alpha}_s$ between symbolic processes, defined in Figure 2, where α ranges over the following alphabet of *symbolic actions*:

1. *symbolic input actions* $X(Y)$ where X and Y are second-order variables, modelling public inputs as in the concrete semantics except that the attacker recipes are replaced by the two placeholders X, Y ;
2. *symbolic output actions* $\bar{X}\langle ax_i \rangle$ that follow the same logic;
3. the *unobservable action* τ which has the exact same role as in the concrete semantics.

Before we define the semantics let us explain how we handle conditionals. First of all we recall our convention to interpret substitutions as sets of equalities, that is, the positive branch

of “if $u = v$ then P else Q ” will add one mgu of u and v modulo theory to E^1 . Regarding the negative branch, we want to add a constraint that is satisfied *iff* u and v are not equal modulo theory. We write it $\neg mgu_{\mathcal{R}}(u =^? v)$ and define it as follows:

$$\neg mgu_{\mathcal{R}}(u =^? v) = \bigwedge_{\sigma \in mgu_{\mathcal{R}}(u =^? v)} \forall z_1, \dots, z_n. \bigvee_{x \in vars(u, v)} x \neq^? x\sigma$$

where $\{z_1, \dots, z_n\} = vars(u\sigma, v\sigma) \setminus vars(u, v)$.

If $C = (\Phi, D, E^1)$, $\mu = mgu(E^1) \neq \perp$ and $n = |dom(\Phi)|$:

$$(\{\text{if } u = v \text{ then } P \text{ else } Q\} \cup \mathcal{P}, C) \xrightarrow{\tau}_s (\{P\} \cup \mathcal{P}, (\Phi, D, E^1 \wedge \sigma)) \quad \text{(S-THEN)}$$

if $\sigma \in mgu_{\mathcal{R}}(u\mu =^? v\mu)$

$$(\{\text{if } u = v \text{ then } P \text{ else } Q\} \cup \mathcal{P}, C) \xrightarrow{\tau}_s (\{Q\} \cup \mathcal{P}, (\Phi, D, E^1 \wedge \neg mgu_{\mathcal{R}}(u\mu =^? v\mu))) \quad \text{(S-ELSE)}$$

$$(\{u(x).P\} \cup \mathcal{P}, C) \xrightarrow{Y(X)}_s (\{P\} \cup \mathcal{P}, (\Phi, D \wedge X \vdash^? x \wedge Y \vdash^? y, E^1 \wedge \sigma)) \quad \text{(S-IN)}$$

if $Y:n, X:n$ and y are fresh and $\sigma \in mgu_{\mathcal{R}}(y =^? u\mu)$

$$(\{\bar{u}(v).P\} \cup \mathcal{P}, C) \xrightarrow{\bar{Y}(ax_{n+1})}_s (\{P\} \cup \mathcal{P}, (\Phi \cup \{ax_{n+1} \mapsto v\mu\sigma\}, D \wedge Y \vdash^? y, E^1 \wedge \sigma)) \quad \text{(S-OUT)}$$

if $Y:n$ and y are fresh and $\sigma \in mgu_{\mathcal{R}}(y =^? u\mu \wedge v\mu =^? v\mu)$

$$(\{\bar{u}(v).P, w(x).Q\} \cup \mathcal{P}, C) \xrightarrow{\tau}_s (\{P, Q\{x \mapsto v\mu\sigma\}\} \cup \mathcal{P}, (\Phi, D, E^1 \wedge \sigma)) \quad \text{(S-COMM)}$$

if $\sigma \in mgu_{\mathcal{R}}(u\mu =^? w\mu \wedge v\mu =^? v\mu)$

$$(\{P \mid Q\} \cup \mathcal{P}, C) \xrightarrow{\tau}_s (\{P, Q\} \cup \mathcal{P}, C) \quad \text{(S-PAR)}$$

Figure 2. A symbolic semantics for the applied pi-calculus

The rule (S-IN) adds two deduction facts $X \vdash^? x$ and $Y \vdash^? y$ to D , modelling that the input term and communication channel should be deducible by the adversary; in particular the constraint $\sigma \in mgu_{\mathcal{R}}(y =^? u\mu)$ indicates that the term deduced by Y is effectively the channel u . The rule (S-OUT) essentially follows the same logic, adding a fresh deduction fact and a constraint indicating that the channel is deducible. We assume an implicit alpha renaming of bound variables so that each appear only once in the process: this prevents reference conflicts in D when applying the rule (S-IN). Let us also point out that several rules introduce constraints of the form $mgu_{\mathcal{R}}(u, u)$: we recall that this substitution is not always \top , but is the most general substitution σ ensuring that $u\sigma$ is a message. As in the concrete semantics, a *symbolic trace* is then a finite sequence of transitions

$$(\mathcal{P}_0, C_0) \xrightarrow{\alpha_1}_s \dots \xrightarrow{\alpha_n}_s (\mathcal{P}_n, C_n)$$

which may be referred to as $(\mathcal{P}_0, C_0) \xRightarrow{\text{tr}}_s (\mathcal{P}_n, C_n)$ if tr is obtained by removing the τ 's from the word $\alpha_1 \cdots \alpha_n$. For simplicity the plain process P may be interpreted as the symbolic process $(\{P\}, (\emptyset, \top, \top))$.

EXAMPLE 3.14. We consider again the example of the private authentication protocol. We recall the process of the agent B receiving the communication, writing pk_X, sk_X instead of $pk(sk(X)), sk(X)$, and $t = \text{adec}(x, s)$:

$$\begin{aligned}
 B(s, p, n, r) &= c(x). \\
 &\text{if } \text{snd}(t) = p \text{ then} \\
 &\quad \bar{c}\langle \text{aenc}(\langle \text{fst}(t), n, \text{pk}(s) \rangle, r, p) \rangle \\
 &\text{else } \bar{c}\langle \text{aenc}(n, r, \text{pk}(s)) \rangle
 \end{aligned}$$

and use a frame $\Phi_0 = \{ax_1 \mapsto pk_A, ax_2 \mapsto pk_B, ax_3 \mapsto \text{aenc}(\langle N_A, pk_A \rangle, r_A, pk_B)\}$, containing public keys and the connection request sent by A . We give in Figure 3 a tree of all symbolic executions of B (we only write the constraints added at each step).

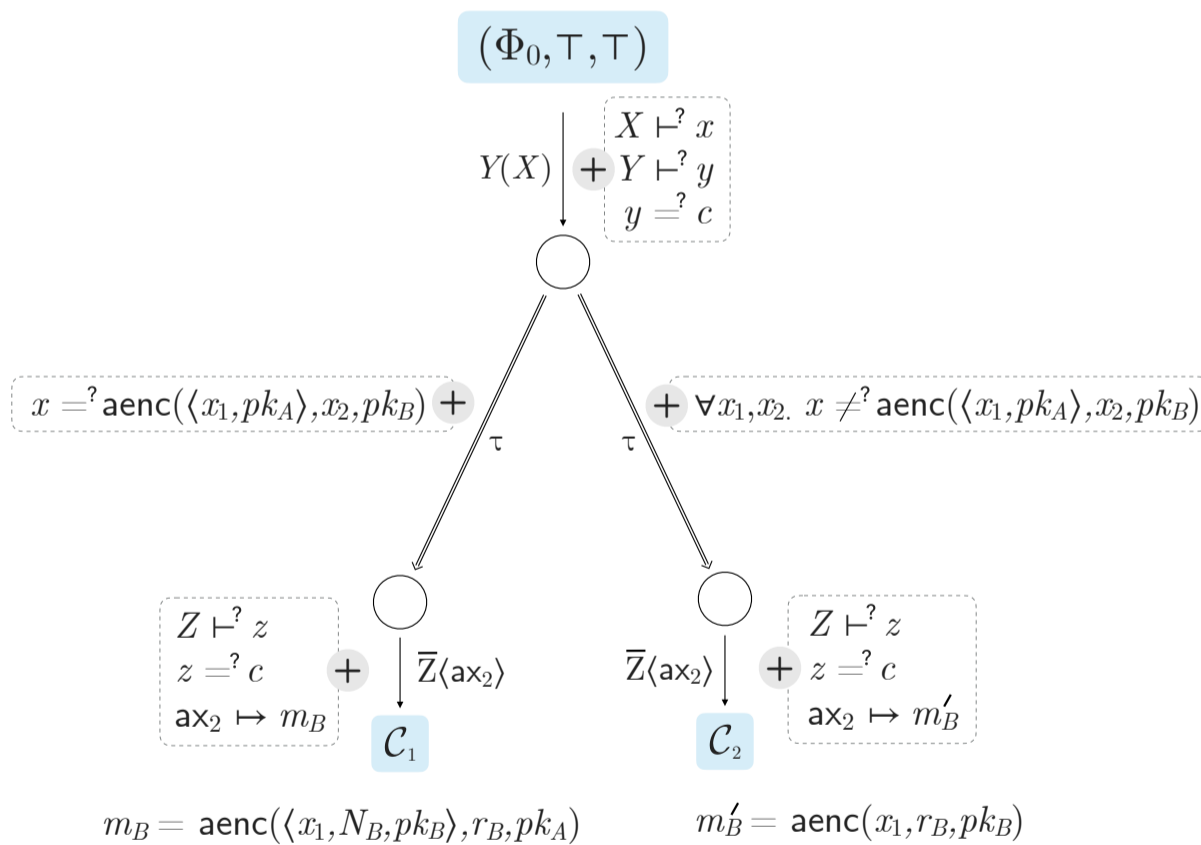


Figure 3. Tree of all constraint systems reachable by executing B symbolically

Intuitively, the branch of the constraint system C_1 abstracts the set of concrete traces where B accepts the connection, and the branch of C_2 those where B refuses it. Typically in the traces of the branch C_1 the attacker forwards the message of A or forges one pretending to be

A; this is formally expressed by the fact that $mgs(C_1) = \{\Sigma_0 \cup \Sigma_{\text{fwd}}, \Sigma_0 \cup \Sigma_{\text{att}}\}$ where:

$$\Sigma_0 = \{Y \mapsto d, Z \mapsto d\} \quad \Sigma_{\text{fwd}} = \{X \mapsto ax_3\} \quad \Sigma_{\text{att}} = \{X \mapsto \text{aenc}(\langle x_1, ax_1 \rangle, x_3, ax_2)\} \quad \blacklozenge$$

Soundness and completeness Similar symbolic semantics have been developed in the context of protocol analysis [15, 29]. The general approach is to abstract the (infinite) set of concrete traces by the finite set of symbolic traces and to study the solutions of the resulting constraint systems. A typical example is that the following statements are equivalent:

1. *Weak secrecy of the term u in P* : for all traces $P \xRightarrow{\text{tr}} (\mathcal{P}, \Phi)$, u is not deducible from Φ
2. for all symbolic traces $P \xRightarrow{\text{tr}}_s (\mathcal{P}, C)$, the system $(\Phi(C), D(C) \wedge X \vdash^? x, E^1(C) \wedge x =^? u)$ has no solution, where $X:n$ and x are fresh, $n = |\text{dom}(\Phi)|$

(Recall that for notational convenience the plain process P may be interpreted as the symbolic process $(\{\!|P|\!\}, (\emptyset, \top, \top))$.) This reduces weak secrecy (for a bounded number of sessions) to the decidability of whether a constraint system has a solution. Similar approaches have been developed in [15, 29] to decide equivalence properties for some classes of processes. They rely on a connection between the symbolic and concrete semantics, under the form of two properties: 1. *soundness*: applying to a symbolic trace a solution of its final constraint system leads to a concrete trace; and 2. *completeness*: all concrete traces are obtained by applying a solution to a symbolic one. They are formalised below, the proof following from a straightforward induction on the length of the traces.

PROPOSITION 3.15 (soundness and completeness of the symbolic semantics). *Let (\mathcal{P}, C) be a symbolic process. Then we have:*

1. *Soundness: for all symbolic traces $(\mathcal{P}, C) \xRightarrow{\text{tr}}_s (Q, C')$ and $(\Sigma, \sigma) \in \text{Sol}(C')$, there exists a concrete trace of the form $(\mathcal{P}\sigma, \Phi(C)\sigma \downarrow) \xRightarrow{\text{tr}_s \Sigma} (Q\sigma, \Phi(C')\sigma \downarrow)$*
2. *Completeness: for all symbolic processes (\mathcal{P}, C) , $(\Sigma, \sigma) \in \text{Sol}(C)$, and for all concrete traces $(\mathcal{P}\sigma, \Phi(C)\sigma \downarrow) \xRightarrow{\text{tr}} (Q, \Phi)$, there exists a symbolic trace $(\mathcal{P}, C) \xRightarrow{\text{tr}'}_s (Q', C')$ and $(\Sigma', \sigma') \in \text{Sol}(C')$ such that $\Sigma \subseteq \Sigma'$, $Q = Q'\sigma'$, $\text{tr} = \text{tr}'\Sigma'$ and $\Phi = \Phi(C')\sigma' \downarrow$.*

3.5 The key tool: the partition tree

To decide trace equivalence and labelled bisimilarity, we introduce the novel notion of a *partition tree* of two bounded processes P and Q . The point is to build a (finite) tree of all symbolic executions of P and Q , grouping into the same nodes intermediary processes as follows:

1. All processes of a same node should have a *common, unique mgs*. Since one symbolic process alone may already have several most general solutions, the node is parametrised by a restricting predicate π on second-order solutions (recall Example 3.13).
2. When applying the mgs of a node to all of the processes it contains (and instantiating the potential remaining variables by fresh distinct constants), the resulting frames are

statically equivalent. Conversely, all reachable symbolic processes that would verify this property should be in the node as well.

A branch of this tree therefore represents the set of all equivalent traces of P and Q taking a given sequence of visible actions. Taking profit of this observation we will show that whenever P and Q are not trace equivalent or labelled bisimilar, a witness of non-equivalence can be exhibited using the tree. Formally its nodes are modelled by *configurations* that consist of sets Γ of symbolic processes sharing a unique mgs and statically equivalent solutions.

DEFINITION 3.16 (configuration). A *configuration* is a pair (Γ, π) where Γ is a set of symbolic processes and π a predicate on second-order substitutions. We also require that:

1. the predicate π is defined on $\text{vars}^2(\Gamma)$, that is, for all Σ , $\pi(\Sigma)$ iff $\pi(\Sigma|_{\text{vars}^2(\Gamma)})$;
2. for all $(\mathcal{P}, C) \in \Gamma$, $|\text{mgs}^\pi(C)| = 1$;
3. for all $(\mathcal{P}_1, C_1), (\mathcal{P}_2, C_2) \in \Gamma$, if $(\Sigma, \sigma_1) \in \text{Sol}^\pi(C_1)$ then there exists σ_2 such that $(\Sigma, \sigma_2) \in \text{Sol}^\pi(C_2)$ and $\Phi(C_1)\sigma_1 \sim \Phi(C_2)\sigma_2$.

The predicate π can typically be described using second-order (dis)equations. We then consider trees with nodes labelled by configurations and edges by visible symbolic actions (i.e., not τ). Given a node n of such a tree, we write $\Gamma(n)$ and $\pi(n)$ the components of the corresponding configuration, and $n \xrightarrow{\alpha} n'$ to express that n' is a child node of n through an edge labelled by the symbolic action α . By definition of a mgs, the points 2 and 3 of Definition 3.16 above ensure that all symbolic processes in $\Gamma(n)$ have the same set of second-order variables, written $\text{vars}^2(n)$, and a common and unique mgs, written $\text{mgs}(n)$.

DEFINITION 3.17 (partition tree). A *partition tree* of two bounded processes P and Q is a tree T whose nodes are labelled by configurations and edges by visible symbolic actions, and that verifies the following properties. First of all $P, Q \in \Gamma(\text{root}(T))$ and $\pi(\text{root}(T)) = \top$, where $\text{root}(T)$ denotes the root node of the tree. Then for all nodes n of T , $(\mathcal{P}, C) \in \Gamma(n)$ and visible symbolic actions α :

1. *Closure by τ -transition:* if $(\mathcal{P}, C) \xrightarrow{\tau}_s (\mathcal{P}', C')$ and $\text{Sol}^{\pi(n)}(C') \neq \emptyset$ then $(\mathcal{P}', C') \in \Gamma(n)$.
2. *All symbolic transitions are reflected in the tree:* if $(\mathcal{P}, C) \xrightarrow{\alpha}_s (\mathcal{P}', C')$ and $(\Sigma, \sigma) \in \text{Sol}^{\pi(n)}(C')$ then there exists an edge $n \xrightarrow{\alpha} n'$ in T such that $(\mathcal{P}', C') \in \Gamma(n')$ and $(\Sigma', \sigma) \in \text{Sol}^{\pi(n')}(C')$ for some Σ' that coincides with Σ on $\text{vars}^2(n)$.

Moreover for all edges $n \xrightarrow{\alpha} n_c$ of T and $(\mathcal{P}_c, C_c) \in \Gamma(n_c)$:

3. *Predicates are refined along branches:* for all Σ , if Σ verifies $\pi(n_c)$ then it verifies $\pi(n)$.
4. *Nodes are maximal:* if $(\Sigma, \sigma) \in \text{Sol}^{\pi(n)}(C)$, $(\Sigma_c, \sigma_c) \in \text{Sol}^{\pi(n_c)}(C_c)$ and $\Sigma \subseteq \Sigma_c$, then $\Gamma(n_c)$ contains all symbolic processes (\mathcal{P}', C') such that $(\mathcal{P}, C) \xrightarrow{\alpha}_s (\mathcal{P}', C')$ and, for some substitution σ' , $(\Sigma_c, \sigma') \in \text{Sol}(C')$ and $\Phi(C_c)\sigma_c \sim \Phi(C')\sigma'$.

The set of partition trees of P and Q is written $\text{PTree}(P, Q)$.

The set $\text{PTree}(P, Q)$ is infinite (at least because arbitrarily many processes can be put in the root configuration) but our decision procedures only require to construct one, arbitrary partition tree. The children n' of a node n represent the sets of processes, grouped w.r.t. static equivalence, reachable by one transition from a process of n . Item 2 ensures that all cases are covered, that is, for all symbolic transitions from n and all solutions Σ , at least one child n' should contain the resulting symbolic process. Note that we do not impose that Σ verifies $\pi(n')$, but that there exists another solution Σ' computing the same first-order terms that does. This more permissive approach will allow us, when generating partition-tree nodes in Section 4, to use families of predicates π that only consider solutions of a certain form (which therefore requires to prove that any deducible term can be computed by a recipe of this form). Item 4 then formalises that the nodes are saturated under static equivalence: if n' is a child of n and a symbolic transition $A \xrightarrow{\alpha}_s B$ from a process $A \in \Gamma(n)$ may result into a process statically equivalent to a process $C \in \Gamma(n')$ then B should be in $\Gamma(n')$ as well.

EXAMPLE 3.18. Let us draw a partition tree corresponding to an anonymity analysis in the private authentication protocol, simplified for readability. We consider the following light version of the role of the process B accepting a connection from an agent X , removing the identification nonces N_A, N_B from the protocol and replacing the decoy message by a fresh name r :

$$B_X = c(x).\text{if } \text{adec}(x, sk_B) = pk_X \text{ then } \bar{c}\langle \text{aenc}(\text{ok}, r, pk_X) \rangle \text{ else } \bar{c}\langle r \rangle$$

We consider a 3-agent scenario (A, B, C) where A has already emitted $\text{aenc}(pk_A, r_A, pk_B)$ to initiate a communication with B . The security property we study is whether the identity of B 's accepted recipient remains anonymous. That is we want to prove $P \approx Q$ where

$$P = C[B_A] \quad Q = C[B_C] \quad C[R] = \bar{c}\langle pk_A \rangle. \bar{c}\langle pk_B \rangle. \bar{c}\langle pk_C \rangle. \bar{c}\langle \text{aenc}(pk_A, r_A, pk_B) \rangle. R$$

The partition tree in Figure 4 has been lightened for readability: if a node contains two symbolic processes A_s, B_s such that $A_s \xrightarrow{\tau}_s B_s$, then A_s is omitted from the node (as it contains less constraints than B_s anyway). The configuration at the root of the tree only contains P and Q . After the four initial outputs of the context C , we reach the constraint system C_0 defined by:

$$\begin{aligned} \Phi(C_0) &= \{ax_1 \mapsto pk_A, ax_2 \mapsto pk_B, ax_3 \mapsto pk_C, ax_4 \mapsto \text{aenc}(pk_A, r_A, pk_B)\} \\ D(C_0) &= X_1 \vdash^? x_1 \wedge X_2 \vdash^? x_2 \wedge X_3 \vdash^? x_3 \wedge X_4 \vdash^? x_4 \\ E^1(C_0) &= x_1 =^? c \wedge x_2 =^? c \wedge x_3 =^? c \wedge x_4 =^? c \end{aligned}$$

The next step is the first one inducing a non-trivial case analysis. This node has four children for the adversary to compute the input $c(x)$: π_1 forwards the message of A , π_2 forges a message

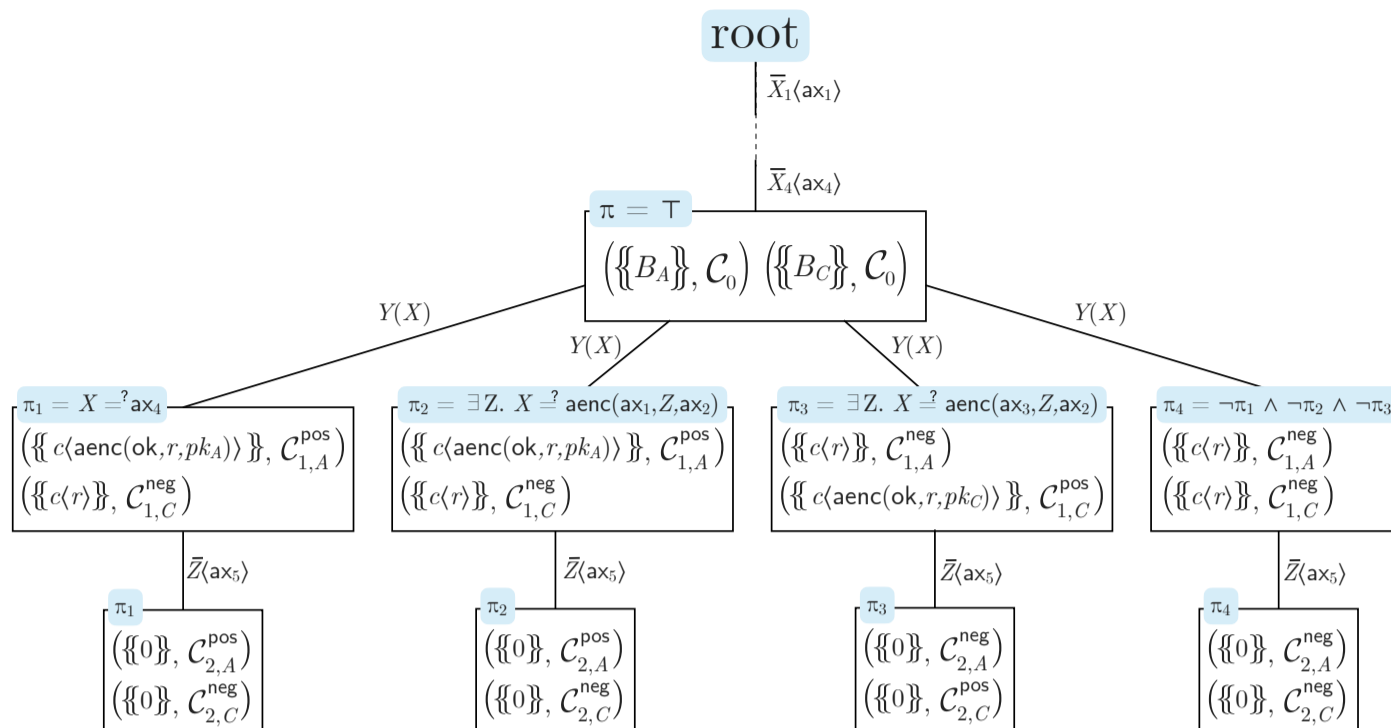


Figure 4. A simplified partition tree of P and Q

pretending it is from A , π_3 forges a message pretending it is from C , π_4 any other case. The choice of these 4 cases is guided by the conditional if $\text{adec}(x, sk_B) = pk_X$ (where $X = A$ or $X = C$) that is evaluated on the input. Choice π_1 results in the positive branch in both B_A and B_C , as it corresponds to an honest execution. Choice π_2 results in choosing the positive branch in B_A and the negative branch in B_C , while π_3 does the opposite. Choice π_4 leads to the negative branch in all cases by construction (as it is the negation of the 3 previous cases).

More precisely we write $\Phi(C_{1,X}^{\text{pos}}) = \Phi(C_{1,X}^{\text{neg}}) = \Phi(C_0)$, $D(C_{1,X}^{\text{pos}}) = D(C_{1,X}^{\text{neg}}) = D(C_0) \wedge Y \vdash? y$ and

$$\begin{aligned} E^1(C_{1,X}^{\text{pos}}) &= E^1(C_0) \wedge y =? c \wedge x =? \text{aenc}(pk_X, x', pk_B) \\ E^1(C_{1,X}^{\text{neg}}) &= E^1(C_0) \wedge y =? c \wedge \forall x'. x \neq? \text{aenc}(pk_X, x', pk_B) \end{aligned}$$

Then the final transitions simply execute the resulting outputs, i.e. $C_{2,X}^s$, $s \in \{\text{pos}, \text{neg}\}$, is obtained by adding $Z \vdash? z$ and $z =? c$ to $C_{1,X}^s$. Since a ciphertext is indistinguishable from a nonce, the two outputs always end up in the same nodes; that is, all leaves contain at least one process originated from P and at least one from Q , which is how we prove trace equivalence. The situation would be different with a rewrite rule such as $\text{test_aenc}(\text{aenc}(x, y, \text{pk}(z))) \rightarrow \text{ok}$; a partition tree of P and Q with this extended rewriting system can be found in Figure 5.

We highlighted the part differing from the previous tree. Essentially some leaf nodes have been split in two due to the enhanced capabilities of the adversary to disprove static equivalence, inducing a violation of trace equivalence. For example the leftmost leaf's mgs is

$$\{X \mapsto ax_4, X_1 \mapsto c, \dots, X_4 \mapsto c, Y \mapsto c, Z \mapsto c\}$$

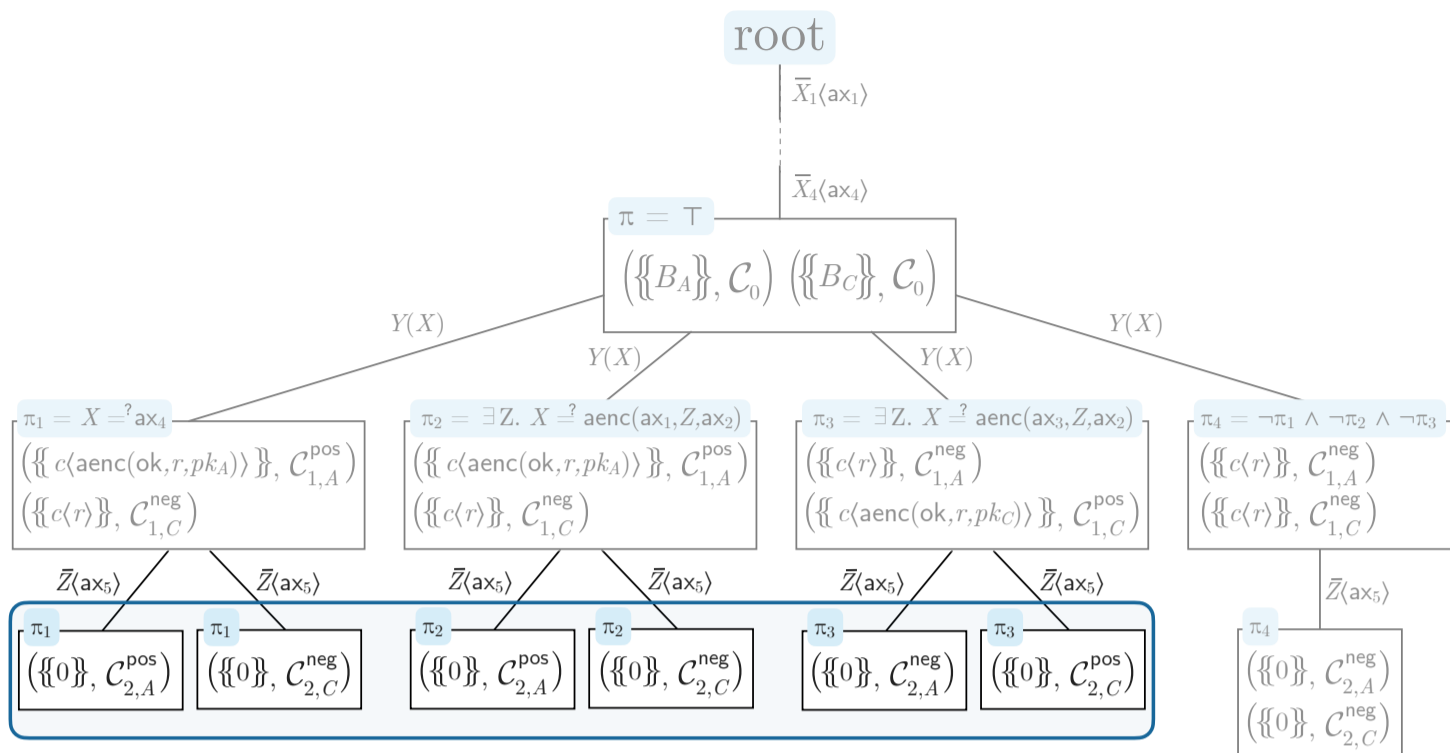


Figure 5. Partition tree with the rewriting system extended with $\text{test_aenc}(\text{aenc}(x, y, \text{pk}(z))) \rightarrow \text{ok}$

which corresponds to an attack trace where the attacker forwards the message of A and observes whether the response of B is a ciphertext, which reveals whether B accepts connections from A or not. \blacklozenge

In the remaining of the section we formalise how to decide trace equivalence and labelled bisimilarity of two processes, given a partition tree and the mgs of each of its nodes. For that we will rely on the following notion of reduction, characterising symbolic traces viewed as branches of a partition tree:

DEFINITION 3.19 (partition-tree trace). Given a partition tree T we write $(\mathcal{P}, C), n \xrightarrow{\alpha}_T (\mathcal{P}', C'), n'$ when:

1. n and n' are nodes of T such that $(\mathcal{P}, C) \in \Gamma(n)$ and $(\mathcal{P}', C') \in \Gamma(n')$; and
2. if $\alpha = \tau$ then $n = n'$, otherwise $n \xrightarrow{\alpha} n'$ and $(\mathcal{P}, C) \xrightarrow{\alpha}_s (\mathcal{P}', C')$.

For convenience this notion is to be understood up to alpha renaming of the variables of the symbolic action α . We write $A_0^s, n_0 \xRightarrow{\text{tr}}_T A_p^s, n_p$ instead of $A_0^s, n_0 \xrightarrow{\alpha_1}_T \cdots \xrightarrow{\alpha_p}_T A_p^s, n_p$ if tr is the word obtained after removing τ symbols from $\alpha_1 \cdots \alpha_p$. If P is a plain process we may also write $P \xRightarrow{\text{tr}}_T A_s, n$ instead of $(\{P\}, (\emptyset, \top, \top), \text{root}(T)) \xRightarrow{\text{tr}}_T A_s, n$.

3.6 Decision procedures for equivalence

In this section, we assume that we managed to compute a partition tree $T \in \text{PTree}(P_1, P_2)$ (in particular, that there exists one). We then describe how to derive a decision procedure for trace equivalence and labelled bisimilarity from T .

Trace equivalence As hinted in our various examples, deciding trace equivalence can be reduced to an analogue notion of equivalence using the (finite) transition relation \xrightarrow{T}^{α} instead of the concrete semantics $\xrightarrow{\alpha}$. This is formalised by the following theorem:

THEOREM 3.20 (partition-tree-based characterisation of trace equivalence). *Whenever $T \in \text{PTree}(P_1, P_2)$, the following points are equivalent:*

1. $P_1 \sqsubseteq_t P_2$
2. for all partition-tree traces $P_1 \xRightarrow{T}^{\text{tr}} (\mathcal{P}_1, C_1), n$, we have $P_2 \xRightarrow{T}^{\text{tr}} (\mathcal{P}_2, C_2), n$

The proof of this result mostly follows from a combination of the soundness and completeness of the symbolic semantics, with two technical lemmas generalising the properties of the partition tree from edges to branches. The detailed statements and proofs can be found in Appendix A.1.

Simulations In the case of trace equivalence, a witness that $A \not\approx_t B$ was simply a trace of A or B that has no equivalent trace in the other process. The case of labelled bisimilarity is however more involved. Using vocabulary borrowed from game theory, the definition of bisimilarity can be seen as a *prover-disprover game*: at each state of the game the disprover chooses a transition from one of the two processes and the prover answers by choosing a transition of the same type from the other process (plus some potential τ -transitions). The disprover wins the game if they manage to reach a state with non-statically-equivalent processes or if the prover cannot answer to one of the moves. A witness of non-equivalence is thus a winning strategy for the disprover. We formalise this below, recalling that if α is an action, we write $\bar{\alpha} = \alpha$ if $\alpha \neq \tau$ and $\bar{\alpha} = \varepsilon$ if $\alpha = \tau$.

DEFINITION 3.21 (witnesses). *A witness of non-bisimilarity w is a set of pairs (A_0, A_1) verifying the following two conditions:*

1. A_0 and A_1 are ground extended processes such that $A_0 \sim A_1$
2. there exists $b \in \{0, 1\}$ and a transition $A_b \xrightarrow{\alpha} A'_b$ such that for all traces $A_{1-b} \xRightarrow{\bar{\alpha}} A'_{1-b}$, either $A'_0 \not\sim A'_1$ or $(A'_0, A'_1) \in w$.

We say that in addition that w is a *witness of non-simulation* if the above two conditions can always be satisfied with $b = 0$. We say that w is a witness for (A_0, A_1) if $(A_0, A_1) \in w$.

Note that the witness can be seen as a relation corresponding to the negation of the definition of bisimilarity (Definition 2.10) minus the static equivalence, i.e. $\not\approx_b \setminus \not\sim$.

PROPOSITION 3.22 (witness-based characterisation of labelled bisimilarity). *If $A_0 \sim A_1$ then:*

1. $A_0 \not\approx_b A_1$ iff there exists a witness of non-bisimilarity w for (A_0, A_1)

2. $A_0 \not\sqsubseteq_s A_1$ iff there exists a witness of non-simulation w for (A_0, A_1)

PROOF. We only give the proof in the case of \approx_b , as the proof for \sqsubseteq_s is analogue. First, we observe that $A_0 \not\approx_b A_1$ iff there exists a binary relation \mathcal{S} on ground extended processes such that $A_0 \mathcal{S} A_1$ and, for all $(B_0, B_1) \in \mathcal{S}$, either 1. $B_0 \not\approx B_1$, or 2. there exists $b \in \{0, 1\}$ and a transition $B_b \xrightarrow{\alpha} B'_b$ such that for all traces $B_{1-b} \xRightarrow{\bar{\alpha}} B'_{1-b}$, $B'_0 \mathcal{S} B'_1$. Let us call such a relation \mathcal{S} a *labelled attack on (A_0, A_1)* . Since processes are bounded there exist no infinite sequences of transitions and for all A, B , $A \not\approx_b B$ therefore straightforwardly rephrases to the existence of a labelled attack \mathcal{S} such that $A \mathcal{S} B$. It then suffices to observe that

1. If \mathcal{S} is a labelled attack on (A_0, A_1) then $\mathcal{S} \setminus \not\approx$ is a witness for (A_0, A_1) .
2. If w is a witness for (A_0, A_1) then $w \cup \not\approx$ is a labelled attack on (A_0, A_1) . ■

We now define a symbolic variant of the notion of witness that can be constructed within a partition tree T . In essence, a symbolic witness may be seen as a winning strategy for the disprover in a bisimulation game limited to the finite transition relation \rightarrow_T .

DEFINITION 3.23 (symbolic witnesses). A *symbolic witness of non bisimilarity* w_s w.r.t. a partition tree T is a finite tree whose nodes N are labelled by tuples (A_0, n) or (A_0, A_1, n) with n a node of T and $A_0, A_1 \in \Gamma(n)$. We also require that if N is labelled (A_0, A_1, n) , there exist $b \in \{0, 1\}$ and a transition $A_b, n \xrightarrow{\alpha} A'_b, n'$ (possibly $\alpha = \tau$) such that:

1. If A_{1-b} is not reducible by $\xRightarrow{\bar{\alpha}}_T$ then N has a unique child labelled (A'_b, n') ;
2. otherwise the children of N are the nodes labelled (A'_0, A'_1, n') , $A_{1-b}, n \xRightarrow{\bar{\alpha}}_T A'_{1-b}, n'$.

We say that w_s is a *witness of non-simulation* if the above two conditions can always be satisfied with $b = 0$. We say that w_s is a *symbolic witness for (A_0, A_1, n)* when $root(w_s)$ is labelled by (A_0, A_1, n) .

However purely symbolic witnesses do not exhibit consistent proofs of non-equivalence in general. Indeed, while a concrete execution fixes the effective value of an input x at the moment it is performed, a symbolic execution records constraints on x all along the execution. Rephrasing, the symbolic semantics puts the prover at a disadvantage in the game, since they have to answer to the disprover's input actions without knowing the values of the input terms. Symbolic witnesses inducing invalid winning strategies for the disprover will be discarded by their absence of *solutions* in the following sense:

DEFINITION 3.24 (solution of a symbolic witness). Let w_s be a symbolic witness. A *solution* of w_s is a function f_{sol} that maps nodes of w_s to ground second-order substitutions such that for all nodes N labelled (A_0, n) or (A_0, A_1, n) ,

1. for all $A_b = (\mathcal{P}, C)$, $(f_{sol}(N), \sigma) \in Sol^{\pi(n)}(C)$ for some σ ;

2. for all children nodes N_1, N_2 of N , $f_{\text{sol}}(N) \subseteq f_{\text{sol}}(N_1) = f_{\text{sol}}(N_2)$.

We denote $\text{Sol}(w_s)$ the set of solutions of w_s .

THEOREM 3.25 (partition-tree-based characterisation of labelled bisimilarity). *If $T \in \text{PTree}(P_1, P_2)$:*

1. $P_1 \approx_b P_2$ iff for all symbolic witnesses of non-bisimilarity w_s for $(P_1, P_2, \text{root}(T))$, we have $\text{Sol}(w_s) = \emptyset$
2. $P_1 \sqsubseteq_s P_2$ iff for all symbolic witnesses of non-simulation w_s for $(P_1, P_2, \text{root}(T))$, we have $\text{Sol}(w_s) = \emptyset$

The proof, although technical, simply connects the symbolic witnesses to concrete ones using the soundness and completeness of the symbolic semantics as well as the properties of the partition tree, following similar ideas as the analogue proof for trace equivalence. The detailed proof can be found in Appendix A.2.

Assuming one has computed a partition tree $T \in \text{PTree}(P_1, P_2)$ and the mgs of each of its nodes, since there are finitely-many possible symbolic witnesses, Theorem 3.25 yields a decision procedure for the labelled bisimilarity of P_1 and P_2 provided one can decide whether a given symbolic witness has a solution. For that we rely on a simple, bottom-up unification of the mgs' appearing in the witness; details can be found in Section 5.4 where we study more precisely the complexity of partition-tree-based decision procedures.

3.7 Generating partition trees (with a constraint-solving oracle)

In this section we detail the skeleton of the procedure for computing a partition tree of two plain processes P_1 and P_2 . The description is modular in that most of the technical details, in particular the modelling of the node predicates and how we obtain the expected properties of the tree, are abstracted by a *constraint-solving oracle* that we detail in the next sections. This section should therefore be seen as the overview of the whole algorithm for deciding equivalence properties, which gives enough insight to discuss our implementation.

The algorithm generates the nodes of the tree top-down, that is, from the root to the leaves. We outline the procedure in Figure 6.

Let us now describe the algorithm to compute $T \in \text{PTree}(P_1, P_2)$ in more details, up to the technical developments detailed in the next sections.

1. First, we initiate a root containing P_1 and P_2 and saturate the configuration by τ transitions. That is, we consider the set of symbolic processes

$$\Gamma(\text{root}(T)) = \left\{ (\mathcal{P}, C) \mid P_i \xrightarrow{\tau}_s (\mathcal{P}, C), i \in \{1, 2\}, \text{Sol}(C) \neq \emptyset \right\}$$

Note that the constraint systems C involved in this definition do not contain deduction facts, which makes the decision of the emptiness of $\text{Sol}(C)$ relatively straightforward.

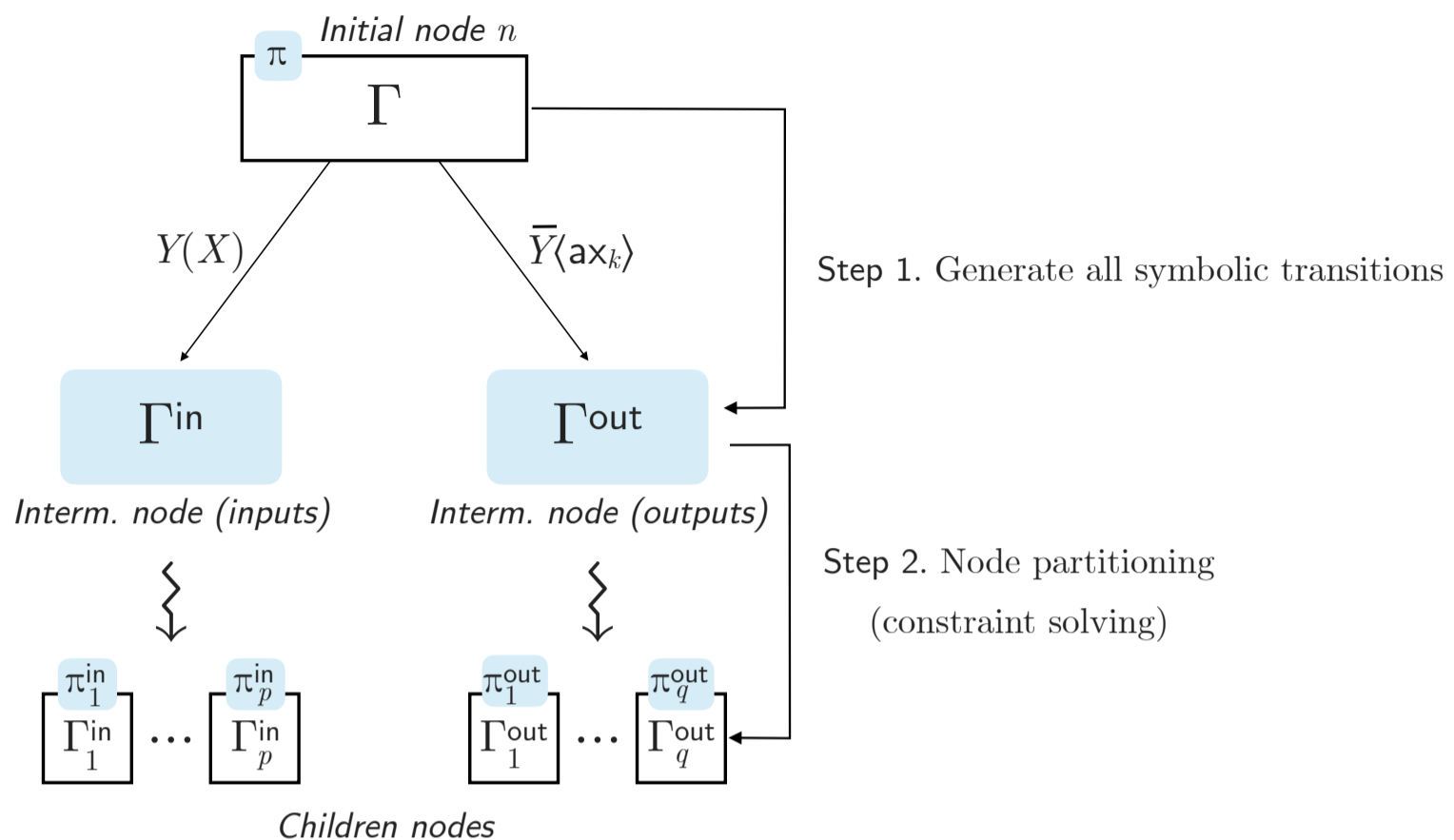


Figure 6. Computing the subtree of a partition tree rooted in a node n

Using the terminology of the later Section 4, using *simplification rules* permits to put the constraints into a simple form where the existence of a solution is trivial to decide.

- Then let us assume we already constructed a node n of the tree using this algorithm, in particular the corresponding configuration $(\Gamma(n), \pi(n))$. To compute the children of n we first enumerate all symbolic transitions from processes of $\Gamma(n)$, separating input and output actions. That is, we compute the two sets

$$\Gamma^{\text{in}} = \left\{ B \mid A \in \Gamma(n), A \xrightarrow{Y(X)}_s B \right\} \quad \Gamma^{\text{out}} = \left\{ B \mid A \in \Gamma(n), A \xrightarrow{\bar{Y}\langle ax_p \rangle}_s B \right\}$$

- Γ^{in} and Γ^{out} are two intermediary sets that do not satisfy yet the father-child properties of the partition tree. For that we use a constraint-solving algorithm detailed in Chapter 4 (*simplification rules* again, but also *case distinction rules*) that will partition Γ^{in} and Γ^{out} to gather symbolic processes with statically-equivalent solutions and remove those with no solutions. This constraint solving results into a sequence of configurations

$$(\Gamma_1^{\text{in}}, \pi_1^{\text{in}}), \dots, (\Gamma_p^{\text{in}}, \pi_p^{\text{in}}) \quad (\Gamma_1^{\text{out}}, \pi_1^{\text{out}}), \dots, (\Gamma_q^{\text{out}}, \pi_q^{\text{out}})$$

that will label the children of n . The procedure is then carried out recursively from these child nodes until no more symbolic transitions are available.

In Section 4 we detail the missing parts of this procedure that take the form of constraint-solving rules, in the context of constructor-destructor subterm convergent theories. Note that the approach is modular in that the proofs we have carried so far are independent of the assumptions on the rewriting system: generalising the results of Section 4 will automatically result in the decidability of trace equivalence and labelled bisimilarity of bounded processes for the extended class of theories.

3.8 Implementation and performances

The DeepSec prover Building on the procedure's structure described above and the internal solver developed in the next sections, we have implemented a prototype in OCaml, called DEEPSEC (DEciding Equivalence Properties in SECurity protocols). The user specifies a rewriting system (that is checked to be constructor-destructor and subterm convergent by the tool), two bounded processes, and the tool verifies whether they are trace equivalent. If not, a concrete attack trace is returned in a dedicated graphical interface; we refer to the DEEPSEC's website for development credits, tutorials and details on practical usage [35]:

<https://deepsec-prover.github.io/>

The tool's specification language implements the grammar presented in Section 2, including some syntax extensions for non-deterministic choice, private function symbols, a restricted form of patterned let bindings, as well as bounded replication $!^n P$ defined as n parallel copies of P . These additional primitives should mostly be seen as syntactic sugar, although the native integration allowed specific optimisations compared to encodings within the initial calculus. The syntax and structure of DEEPSEC's input files are similar to the widely used PROVERIF tool [22] to make it easier for new users to discover and use it.

Trace equivalence vs (bi)simulation The tool currently only implements the trace equivalence procedure as it is rather efficient. Following Theorem 3.20, the procedure for checking trace equivalence between P_1 and P_2 consists in generating the partition tree and checking that each node contains symbolic constraint systems both from P_1 and P_2 . As different branches of the partition tree are independent from one another, the implementation only requires to store in memory the current branch that is being verified, instead of the whole partition tree. On the other hand, the procedure for checking (bi)simulation both requires to compute and store in memory the full partition tree. In addition, the procedure also requires guessing a symbolic witness, which would be extremely inefficient. A natural follow up to our work would be to explore ways of effectively implementing the decision procedure for (bi)simulation that would avoid these two main hurdles.

Partial order reductions The tool also implements *partial order reductions* (POR), an optimisation technique for protocol analysis developed by Baelde et al. [10]. The basic idea is to discard part of the state space that is redundant but this optimisation is only sound when processes are *action-determinate*, as defined in [10]. Although we omit here the definition of determinacy for simplicity, let us mention that not using private channels and assigning a different channel name to each parallel process is a simple, syntactic way to ensure this property. This is however not always possible—typically when looking at some anonymity or unlinkability properties. Typically, the private authentication protocol used as a running example can be modelled as a determinate process, but not the Helios and BAC protocols (due to private channels or because this introduces artificial violations of the equivalence property).

In practice, DEEPSEC automatically detects action-determinate processes and activates the POR, which drastically reduces the number of symbolic executions that need to be considered. We also go further and allow to verify a refined equivalence, *equivalence by session*, that allows to use similar POR techniques without the restriction to determinate processes. This contribution is however out of the scope of this article; details can be found in [32] and our experimental results presented below only include the base POR of [10].

Distributing the computation The main task of DEEPSEC is to generate a partition tree and, as we explained, this is done using a top-down approach. This task can be distributed as computing a given node of the tree can be done independently of its sibling nodes. However, some engineering is needed to avoid heavy communication overhead due to task scheduling. Indeed, the partition tree is not a balanced tree and we do not know in advance which branches will be larger than others. Because of this, we do not directly compute and return the children of each node in the most straightforward manner, but proceed in two steps:

1. We start with a breadth-first generation of the partition tree. The number of pending nodes will gradually grow until eventually exceeding a threshold parameter n .
2. Each available core focuses on one of these nodes, computes the whole subtree rooted in this node in a depth-first manner and, when this the task is completed, is assigned to a new node until none remain.

If some cores become idle for too long in Step 2 (because the number of cores exceeds the number of non-completed nodes), we perform a *new round*, that is, we interrupt the working nodes and restart this two-step procedure on incomplete nodes. Although doing so wastes some proof work, this improves performances for particularly unbalanced trees. Note that parallelisation is also supported by other automated analysers such as AKISS [24], but DEEPSEC goes one step further as it is able to distribute the computation not only on multiple cores of a given machine but also clusters of computers.

Benchmarks We performed extensive benchmarks to compare DEEPSEC against other tools that verify equivalence properties for a bounded number of sessions: AKISS [24], APTE [25], SATEQUIV [43] and SPEC [60]. Experiments are carried out on Intel Xeon 3.10GHz cores, with 40Go of memory. We distributed the computation on 20 cores for AKISS and DEEPSEC as they support parallelisation—unlike the others which therefore use a single core. The results are summarised in Table 2 with the following symbol conventions:

- ✓ analysis terminates and equivalence holds
- ⚡ analysis terminates and an attack is found
- OM analysis aborted due to memory overflow (Out of Memory)
- 🕒 analysis aborted due to timeout (12 hours)
- ✗ the tool is not expressive enough to analyse the protocol

We first analysed strong secrecy and anonymity for several classical authentication protocols. The DEEPSEC tool clearly outperforms AKISS, APTE, and SPEC. The SATEQUIV tool becomes more efficient, when the number of sessions significantly increases.

To put more emphasis on the broad scope we also include analyses of unlinkability and anonymity properties for a number of other protocols. This includes the Private authentication protocol used as a running example, BAC [51] and the Helios voting protocol [5]. In addition we study a simplified version of the AKA protocol deployed in 3G telephony networks without XOR [8], the Passive Authentication protocol implemented in the European passport [51], as well as the Prêt-à-Voter protocol (PaV) [57]. Note that, while PaV is a priori in the scope of AKISS, it failed to produce a proof: AKISS only approximates trace equivalence of non-determinate processes and finds a false attack here. Finally we note that BAC, PaV and Helios protocols are not action-determinate and therefore do not benefit from the POR optimisation, which explains the much higher verification times when increasing the sessions. Nevertheless, as exemplified by some examples, attacks may be found very efficiently, as it generally does not require to explore the entire state space.

4. Generation of the partition tree

In the previous sections, we detailed how to use the partition tree to derive decision procedures for equivalence properties. We describe in this section a constraint solving procedure that may be used to generate one in practice.

4.1 Extended constraint systems

In order to carry out the constraint solving required to construct the partition tree, we extend constraint systems with components allowing to reason more finely about the *attacker's knowl-*

Protocol (# of roles)		AKISS	APTE	SPEC	SATEQUIV	DEESEC
Strong secrecy	Denning-Sacco	3	✓ <1s	✓ <1s	✓ 11s	✓ <1s
		6	✓ <1s	✓ 1s	⊗	✓ <1s
		7	✓ 6s	✓ 3s		✓ <1s
		10	⊗	✓ 9m49		✓ <1s
		12		🕒		✓ <1s
		29				✓ <1s
	Wide Mouth Frog	3	✓ <1s	✓ <1s	✓ 5s	✓ <1s
		6	✓ <1s	✓ <1s	✓ 1h11m	✓ <1s
		7	✓ <1s	✓ 1s	⊗	✓ <1s
		10	✓ 10s	✓ 3m35		✓ <1s
		12	✓ 22m16s	🕒		✓ <1s
		14	⊗			✓ <1s
	Yahalom-Lowe	3	✓ <1s	✓ <1s	✓ 7s	✓ <1s
		6	✓ 2s	✓ 41s	⊗	✓ <1s
		7	✓ 42s	✓ 34m38s		✓ 1s
		10	⊗	🕒		✓ 1s
		12				✓ 4s
		14				✓ 7s
17					✓ 12s	
Anonymity	Private Authentication	2	✓ <1s	✓ <1s		✓ <1s
		4	✓ <1s	✓ 1s		✓ <1s
		6	✓ 21s	✓ 4m18s	×	×
		8	⊗	🕒		✓ 1s
		10				✓ 2s
		15				✓ 32s
Unlinkability	3G-AKA	4	✓ 1m35s	✓ 1h23m	×	×
		6	⊗	🕒		✓ 2s
	Passive Authentication	4	✓ <1s	✓ 1s		✓ <1s
		6	✓ 2m15s	✓ 1m27s		✓ <1s
		7	✓ 1h40m	✓ 1m44s	×	×
		9	🕒	✓ 2h08m		✓ <1s
		15		🕒		✓ 9s
	21				✓ 15s	
	BAC	4	⊗	⚡ 38m56s	×	×
		6		🕒		🕒
Ballot privacy	Prêt-à-Voter	6	×	×	×	✓ 2s
	Helios Vanilla	6	⚡ 47s	⚡ <1s	×	×
	Helios ZKP (vote swap)	10				✓ <1s
		11	⊗	×	×	×
12				✓ 1h 38m		

Table 2. Performances of DeepSec (20 cores) against other protocol analysers

edge. The notion of solution of constraint system is also extended to capture their expected properties.

4.1.1 Knowledge base and formulas

New constraints From now on we assume the existence of a rewriting system \mathcal{R} that is constructor-destructor and subterm convergent (we recall that the results of the previous sections did not rely on this assumption). All our definitions, lemmas and theorems will thus implicitly depend on this rewriting system. We introduce an extension of constraint systems with second-order constraints that serve key roles in the generation of the partition tree:

- ▷ *Giving a finite representation of the deductive capabilities of the attacker.*

This takes the form of a *knowledge base* K which is a finite set of deduction facts. By relying on subterm convergence among others, our procedure will ensure that a term u is deducible *iff* it can be deduced by applying constructor symbols to deduction facts of K , which makes deducibility easily decidable due to the constructor-destructor property. In particular we will only consider solutions that compute terms using entries of K in this restricted manner.

- ▷ *Giving a finite representation of the distinguishing capabilities of the attacker.*

This takes the form of a set of *formulas* F that is, in short, a finite representation of the term equalities that hold in the current frame. In particular static equivalence will be characterisable only from the formulas of F .

- ▷ *Recording the constraints imposed on second-order solutions during the constraint solving.*

When computing most general solutions or performing case analyses on the form of solutions, we track the resulting effect on second-order solutions in a set E^2 that is the second-order analogue of E^1 . This is mostly how we model the predicates π that appear in the configurations in partition trees (Definition 3.16).

More formally we consider, in addition to deduction facts and second-order equations, a new atomic second-order constraint, *equality facts* $\xi =_f^? \zeta$, ξ and ζ second-order terms. Unlike second-order equations that model syntactic equalities, equality facts capture equalities modulo theory, that is, the fact that ξ and ζ deduce the same first-order term. Concretely we extend the relation \models (Section 3.3) with

$$(\Phi, \Sigma, \sigma) \models \xi =_f^? \zeta \quad \text{iff} \quad \xi \Sigma \Phi \sigma \downarrow = \zeta \Sigma \Phi \sigma \downarrow$$

We now define the constraints that are typically put in the set F .

DEFINITION 4.1 (deduction formula, equality formula). A *deduction* (resp. *equality*) *formula* is a constraint of the form $\forall S. H \Leftarrow (C_1 \wedge \dots \wedge C_n)$:

1. S is a set of (both first-order and second-order) variables;

2. H is a deduction fact (resp. an equality fact);
3. for all $i \in \{1, \dots, n\}$, C_i is either a deduction fact of the form $X \vdash^? t$, $X \in \mathcal{X}^2$, or a first-order syntactic equation $u =^? v$.

A formula ψ is called *solved* when it contains no hypotheses, i.e., $\psi = (\forall \emptyset. H \Leftarrow \top) = H$. Given a formula $\psi = \forall S. H \Leftarrow \varphi$, we denote by $\text{hyp}(\psi)$ the set of the syntactic equations appearing in the hypotheses φ , and by $D(\psi)$ the set of deduction facts in φ .

Intuitively, a formula captures a deduction or comparison that the attacker may perform and the premisses C_1, \dots, C_n express conditions under which this is possible. Typically if the attacker observed a ciphertext $\text{senc}(m, r, k)$ (bound to an axiom ax), we may express the deducibility of m through the formula

$$\forall X. \text{sdec}(\text{ax}, X) \vdash^? m \Leftarrow X \vdash^? k$$

Another example is the following formula that expresses the tautology that two recipes deducing the same term should be equal in the sense of an equality fact:

$$\forall X, Y, z. X =_f^? Y \Leftarrow (X \vdash^? z \wedge Y \vdash^? z)$$

This formula will serve as a generic placeholder when computing equality formulas during the constraint solving, that is, we will always add equality formulas obtained by substituting variables in the above formula. Although we consider arbitrary formulas such as the above two during the computation of the partition tree, note that only formulas of a certain shape will eventually be added in the set F recording the attacker's distinguishing capabilities. We give more details about the invariants of the procedure in Appendix B.1, but we can mention for example that the formulas effectively recorded in F will be of the form $H \Leftarrow \varphi$, i.e., there are no universally-quantified variables, and φ only contains first-order equations.

Extended constraint systems We now formalise how we extend constraint systems to store the knowledge base, formulas, and to capture restrictions on the form of solutions.

DEFINITION 4.2 (extended constraint system). A tuple $C^e = (\Phi, D, E^1, E^2, K, F)$ is called an *extended constraint system* where:

1. (Φ, D, E^1) is a constraint system, although more general in that D may contain constraints of the form $X \vdash^? u$ or $\forall X. X \not\vdash^? u$ where u may be an arbitrary constructor term;
2. E^2 is a set of second-order equations and constraints of the form $\forall Y_1, \dots, Y_k. \bigvee_{j=1}^p \xi_j \neq^? \zeta_j$
3. K is a set of deduction facts;
4. F is a set of deduction and equality formulas.

As explained earlier, the set E^2 gathers constraints to be satisfied by the second-order solutions of the system, K is a finite representation of the attacker knowledge, and F characterises

the attacker capabilities to deduce and compare terms modulo theory. In particular the set E^2 contains additional constraints to be satisfied by solutions while K and F are valid formulas that characterise potential attacker actions. For example, the (unsolved) deduction formulas in F reason about potentially deducible terms: when such formula contains premisses, the procedure will perform a case analysis to distinguish cases where the hypotheses hold or not, leading to solved or trivial formulas, respectively. When a solved deduction formula is obtained this way, we add it to the knowledge base K if u is not already deducible from it.

4.1.2 (Most general) solutions

We now define how the notion of solutions is lifted to extended constraint systems and how this embeds the predicates π used in the definition of partition-tree configurations. The definition of a solution (Σ, σ) of C^e follows three guidelines: 1. it should be a solution in the usual sense and satisfy $E^2(C^e)$; 2. the set of formulas $F(C^e)$ plays no role in the definition of solutions: we will only prove invariants that this set verifies during our specific constraint-solving procedure (see Appendix B, Section B.1); and 3. all recipes used in the solution should have been constructed from the knowledge base $K(C^e)$, *uniformly* (that is, a same first-order term should not be deduced by different recipes in the solution). In particular this requires a notion of *consequence*, indicating that a recipe can be deduced from the knowledge base.

DEFINITION 4.3 (consequence). We define the set of *consequences* of a set of deduction facts S , denoted $\text{Conseq}(S)$, as the set of pairs $(C[\xi_1, \dots, \xi_n], C[u_1, \dots, u_n])$ where C is a context built using $\mathcal{F}_c \cup \mathcal{F}_0$ and for all $i \in \{1, \dots, n\}$, $\xi_i \vdash^? u_i \in S$. We write $\xi \in \text{Conseq}(S)$ if $\exists t. (\xi, t) \in \text{Conseq}(S)$.

We recall that by definition a deduction fact never has a constructor function symbol at its root (Definition 3.4): in particular if $\xi \in \text{Conseq}(S)$, the context C in the above definition is unique. Writing $\xi = C[\xi_1, \dots, \xi_n]$ it is therefore possible to define unambiguously the set of *consequential subterms* of ξ

$$st_c(\xi, S) = \{\xi|_p \mid p \text{ position of } C\}$$

If R is a set of recipes we write $st_c(R, S) = \bigcup_{\xi \in R} st_c(\xi, S)$. From this we can define solutions of extended constraint systems.

DEFINITION 4.4 (solution of an extended constraint system). A pair of substitutions (Σ, σ) is a solution of $(\Phi, D, E^1, E^2, K, F)$ if $(\Phi, \Sigma, \sigma) \models D \wedge E^1 \wedge E^2$ and the following two properties hold:

1. *K-Basis*: for all $\xi \in st_c(\text{img}(\Sigma) \cup K\Sigma)$, $\text{msg}(\xi\Phi\sigma)$ and $(\xi, \xi\Phi\sigma\downarrow) \in \text{Conseq}(K\Sigma\sigma)$
2. *Uniformity*: for all $\xi, \xi' \in st_c(\text{img}(\Sigma), K\Sigma)$, $\xi\Phi\sigma\downarrow = \xi'\Phi\sigma\downarrow$ implies $\xi = \xi'$.

The set of solutions of C^e is written $Sol(C^e)$ and C^e is *satisfiable* if $Sol(C^e) \neq \emptyset$. We will denote by \perp an unsatisfiable extended constraint system. The notion of most general solution of C^e is adapted in a straightforward way from the analogue for regular constraint systems.

Intuitively when computing a node n of a partition tree, the extended constraint systems represent the predicate $\pi(n)$: it will be defined so that given $(\mathcal{P}, C) \in \Gamma(n)$ attached with C^e , we have $Sol^{\pi(n)}(C) = Sol(C^e)$ (up to domain restriction). We detail this in Sections 4.1.3 and 4.5.

EXAMPLE 4.5. Consider the extended constraint system C^e defined by

$$\begin{aligned} \Phi &= \{ax_1 \mapsto \langle k, x \rangle\} & D &= X:0 \vdash^? x \wedge Y:1 \vdash^? y & E^1 &= y =^? x & E^2 &= \top & K &= ax_1 \vdash^? \langle k, x \rangle \\ F &= K \wedge fst(ax_1) \vdash^? k \wedge snd(ax_1) \vdash^? x \wedge X =^?_f snd(ax_1) \end{aligned}$$

This system involves an adversarial input x computable from an empty frame, which produces in response an output of $\langle k, x \rangle$ for some name k , and then the adversary inputs again $y = x$. The set F , although not impacting the notion of solution, characterises here all successful operations that the attacker may perform in this situation: applying destructors to the term bound to ax_1 and observe that X and $snd(ax_1)$ deduce the same term.

We have for example $(\langle X, ax_1 \rangle, \langle x, \langle k, x \rangle \rangle) \in \text{Conseq}(K \cup D)$. However the knowledge base is not *saturated* in the sense that there are deducible terms u , for example $u = k$, such that there exist no recipes ξ such that $(\xi, u) \in \text{Conseq}(K \cup D)$. In our procedure, the saturation is done by adding to K all destructor applications that result into a non-consequence term. A saturated version of the constraint system would be

$$C_s^e = C^e [K \mapsto K \wedge fst(ax_1) \vdash^? k]$$

Note that adding the deduction fact $snd(ax_1) \vdash^? x$ to the knowledge base is possible but redundant since x is already deducible from X . The saturation ensures that for all (Σ, σ) satisfying $D(C_s^e) \wedge E^1(C_s^e) \wedge E^2(C_s^e)$, there exists Σ' such that $(\Sigma', \sigma) \in Sol(C_s^e)$, meaning that the requirement that solutions verify K -basis can always be satisfied (which is key for satisfying the requirement that all symbolic transitions are reflected in the partition tree, recall Item 2 of Definition 3.17). Let us then consider

$$\Sigma = \{X \mapsto a, Y \mapsto snd(ax_1)\} \quad \Sigma' = \{X \mapsto a, Y \mapsto a\} \quad \sigma = \{x \mapsto a, y \mapsto a\}$$

Both (Σ, σ) and (Σ', σ) are solutions of the regular constraint system $(\Phi(C_s^e), D(C_s^e), E^1(C_s^e))$, but only (Σ', σ) is a solution of C^e . This is because Σ does not verify uniformity: two different recipes a and $snd(ax_1)$ are used to deduce the same first-order term a . More generally we have $mgs(C_s^e) = \{Y \mapsto X\}$. To obtain this result, the constraint-solving procedure for computing mgs , detailed in Section 4.2, will observe that X and Y deduce the same term and should therefore be unified to satisfy uniformity. A second-order equation $X =^? Y$ is thus added in E^2 , whose mgu is then the expected most general solution. \blacklozenge

REMARK 4.6 (uniformity and complexity). In some sense enforcing that solutions are uniform ensures their minimality in terms of DAG size, by forcing identical recipes to be reused as much as possible when constructing the solution. This will be key for the complexity of our decision procedure, see Section 5.2.

4.1.3 Constraint Solving: the basics

Now we give details about the organisation of our constraint solver, detailed and proved correct in the next sections. As explained in Section 4.1.2, the goal of extended constraint system is to carry additional, structural information about solutions in a node n , thus playing the role of the predicate $\pi(n)$. More formally the procedure operates on:

DEFINITION 4.7 (extended symbolic process, vector). An *extended symbolic process* is a tuple (\mathcal{P}, C, C^e) where (\mathcal{P}, C) is a symbolic process and C^e an extended constraint system. We call a *vector* a set of sets of extended symbolic processes $\mathbb{S} = \{\Gamma_1, \dots, \Gamma_n\}$. Each set Γ_i is called a *component* of \mathbb{S} .

An extended symbolic process (\mathcal{P}, C, C^e) induces a predicate π on the solutions of (\mathcal{P}, C) defined as follows: if $(\Sigma, \sigma) \in \text{Sol}(C)$, then $\pi(\Sigma)$ holds *iff* there exists $(\Sigma', \sigma') \in \text{Sol}(C^e)$ such that $\Sigma \subseteq \Sigma'$ and $\sigma \subseteq \sigma'$. In particular $\text{Sol}^\pi(C) = \text{Sol}(C^e)$ (up to domain restriction). However, we recall that, in the definition of the node of a partition tree (configurations, Definition 3.16), only *one* common predicate π is used for all constraint systems C of the configuration Γ . For consistency, we therefore have to impose conditions ensuring that the predicates π corresponding to each $C \in \Gamma$ are all identical. Given a set of constraint systems Γ such that this property is not verified, the goal of the constraint-solving procedure is thus to refine Γ until obtaining a vector $\mathbb{S} = \{\Gamma_1, \dots, \Gamma_n\}$ such that

1. each component Γ_i can be used to model a partition-tree node, that is, a predicate π_i can be defined as above uniformly across all elements of Γ_i ;
2. the underlying nodes verify the properties of the partition tree w.r.t. their father node Γ .

The procedure takes the form of various reduction relations that are used to refine a set of sets of extended symbolic processes, progressively, until reaching the final vector \mathbb{S} :

1. A set of rules to compute *most general solutions* (Section 4.2).
2. A set of *symbolic rules* (Section 4.3.1) that formalise how to apply symbolic transitions to extended symbolic processes.
3. Various sets of *simplification rules* (Sections 4.2.3, 4.3.2 and 4.3.3) that simplify vectors to remove unsatisfiable systems, or to split components that contain processes with non-statically-equivalent solutions.

4. A set of *case distinction rules* (Section 4.4) that refines the current vector based on case analyses to enforce the various properties of the partition tree (unique mgs in each component, maximal components w.r.t. static equivalence...).

The overall procedure organising the above sets of rules into a complete algorithm to compute a partition tree is then detailed in Section 4.5. This can therefore be seen as the detailed version of the outline provided in Section 3.7. The main arguments for proving the correctness of the computation are also provided in Section 4.5; note however that these are only arguments of *partial correctness*: the termination of the procedure is later justified in Section 5.2.

4.2 Constraint Solving: computing most general solutions

4.2.1 Applying solutions and unifiers

Because solutions Σ may introduce new second-order variables, their applications to a constraint system or a formula is not straightforward. Let for example $C^e = (\Phi, D, E^1, E^2, K, F)$ where a variable $X:k$ is used to deduce a term u , i.e. $(X \vdash^? u) \in D$. Now say we want to consider the scenario where u is computed using a constructor $f/3$ and an entry of the knowledge base $(\xi \vdash^? v) \in K$ as a first argument, that is, we want to apply to C^e :

$$\Sigma = \{X \rightarrow f(\xi, X_1, X_2)\} \quad X_1, X_2 \text{ fresh}$$

The raw application $C^e\Sigma$ has a flawed structure, in particular because the variables X_1 and X_2 would not be bound in the resulting system. To solve this issue we use a custom application mechanism that replaces $X \vdash^? u$ in D by $X_1 \vdash^? x_1, X_2 \vdash^? x_2, x_1, x_2$ fresh, and we add the equality $u =^? f(v, x_1, x_2)$ to E^1 to express the logical link between X and $X\Sigma = f(\xi, X_1, X_2)$.

DEFINITION 4.8 (application of a substitution to an extended constraint system). Let $C^e = (\Phi, D, E^1, E^2, K, F)$ and Σ be a substitution. We write $C^e:\Sigma$ the constraint system:

$$(\Phi, D', E^1 \wedge E_\Sigma, E^2\Sigma \wedge \Sigma|_{\text{vars}^2(C^e)}, K\Sigma, F\Sigma)$$

where $D' = (D \setminus D_{dom}) \cup D_{fresh}$ with the sets of:

1. deduction facts removed by the application of Σ : $D_{dom} = \{Y \vdash^? u \in D \mid Y \in \text{dom}(\Sigma)\}$
2. binding facts: $D_{fresh} = \{Y \vdash^? y \mid Y \in \text{vars}^2(\text{img}(\Sigma|_{\text{vars}^2(C^e)})) \setminus \text{vars}^2(C^e), y \text{ fresh}\}$
3. linking equations: $E_\Sigma = \{u =^? v \mid Y \vdash^? u \in D_{dom}, (Y\Sigma, v) \in \text{Conseq}(K\Sigma \cup D')\}$

By abuse of notation we may write $S:\Sigma$ for $(\mathcal{P}, C, C^e:\Sigma)$ if $S = (\mathcal{P}, C, C^e)$.

We will also use a similar mechanism for applying substitutions to formulas:

DEFINITION 4.9 (application of a substitution to a formula). Let $C^e = (\Phi, D, E^1, E^2, K, F)$, $\psi = \forall S. H \Leftarrow \varphi$ be a formula, and Σ be a substitution. We denote $\psi:(\Sigma, C^e)$ (or $\psi:(\Sigma, S)$) by abuse

of notations if $S = (\mathcal{P}, C, C^e)$ the formula

$$\forall S'. H\Sigma \Leftarrow (D' \wedge \text{hyp}(\psi) \wedge E_\Sigma)$$

where $D' = (D(\psi) \setminus D_{dom}) \cup D_{fresh}$, $S' = (S \setminus \text{dom}(\Sigma)) \cup \text{vars}^1(D_{fresh})$ and:

1. $D_{dom} = \{Y \vdash^? u \in D(\psi) \mid Y \in \text{dom}(\Sigma)\}$
2. $D_{fresh} = \{Y \vdash^? y \mid Y \in \text{vars}^2(\text{img}(\Sigma)) \setminus \text{vars}^2(C, \psi), y \text{ fresh}\}$
3. $E_\Sigma = \{u =^? v \mid Y \vdash^? u \in D_{dom}, (Y\Sigma, v) \in \text{Conseq}(K \cup D \cup D')\}$

4.2.2 Constraint-solving rules

A complete example By definition, the solutions of an extended constraint systems C^e have to verify $K(C^e)$ -basis, which means that in practice we only have to compute solutions constructed by applying constructors to the entries of the knowledge base and D . Besides due to the uniformity requirement we can always unify two recipes that deduce the same first-order term. Putting everything together the most general solutions of an extended constraint system can then be computed with a simple transition system. Let us detail a complete example to illustrate the mechanisms in play, before formalising the corresponding constraint-solving rules.

EXAMPLE 4.10. Given $k, r \in \mathcal{N}$, let us consider a situation where the attacker has observed the output of a hash $h(r)$, then inputs a term x , receives in response a ciphertext $\text{aenc}(k, r, x)$ encrypted with x , and finally inputs a term y that should verify the equation $y =^? \langle k, h(x) \rangle$. This is modelled by the frame $\Phi = \{ax_1 \mapsto h(r), ax_2 \mapsto \text{aenc}(k, r, x)\}$ and the constraints

$$D = X:1 \vdash^? x \wedge Y:2 \vdash^? y \qquad E^1 = y =^? \langle k, h(x) \rangle$$

At this point a saturated knowledge base should contain the two entries of the frame and one recipe indicating that decrypting ax_2 results in obtaining the name k .

$$K = ax_1 \vdash^? h(r) \wedge ax_2 \vdash^? \text{aenc}(k, r, x) \wedge \text{adec}(ax_2, X) \vdash^? k$$

We consider that $E^2 = \top$ and we leave the set of formulas F unspecified since it has no influence on solutions. First of all some *simplification rules* will be applied to propagate the equations on x and y to the whole system; here it will apply $\text{mgu}(E^1)$ to D , resulting in

$$D = X \vdash^? x \wedge Y \vdash^? \langle k, h(x) \rangle$$

The constraint-solving rules detailed in the remaining of this section consider all ways to compute recipes for X and Y from the knowledge base. For each of these recipes two cases arise: either 1. it is picked directly from the knowledge base; or 2. it starts with a constructor symbol. This will correspond to the constraint-solving rules (MGS-RES) and (MGS-CONS), respectively. Finally, to satisfy the uniformity property, the procedure unifies any second-order terms in the

system that deduce the same first-order term (Rule (MGS-CONSEQ)). We keep on refining the case analysis with these three rules, removing branches yielding contradictions, until no more rules are applicable. The resulting systems will either have no solutions, or be in a so-called *solved form* and have $mgu(E^2)$ as a unique mgs. Let us do it for our example:

- ▷ *case 1: the recipe for Y has a constructor symbol at its root (only possible case)*

The constructor in question is necessarily the pair. Therefore we let two fresh second-order variables $Y_1:2, Y_2:2$ and apply the substitution $\{Y \mapsto \langle Y_1, Y_2 \rangle\}$ to the system (in the sense of Definition 4.8). After simplification this leads to the updated second-order constraints:

$$D = X \vdash^? x \wedge Y_1 \vdash^? k \wedge Y_2 \vdash^? h(x) \qquad E^2 = Y =^? \langle Y_1, Y_2 \rangle$$

- ▷ *case 1.1: the recipe for Y_1 is $\text{adec}(ax_2, X)$ from the knowledge base (only possible case)*

We thus apply the substitution $\{Y_1 \mapsto \text{adec}(ax_2, X)\}$, resulting in the updated constraints:

$$D = X \vdash^? x \wedge Y_2 \vdash^? h(x) \qquad E^2 = Y =^? \langle \text{adec}(ax_2, X), Y_2 \rangle \wedge Y_1 =^? \text{adec}(ax_2, X)$$

- ▷ *case 1.1.1: the recipe for Y_2 is the entry ax_1 from the knowledge base*

We therefore apply the substitution $\{Y_2 \mapsto ax_1\}$, resulting in the updated constraints:

$$D = X \vdash^? r \qquad E^2 = Y =^? \langle \text{adec}(ax_2, X), ax_1 \rangle \wedge Y_1 =^? \text{adec}(ax_2, X) \wedge Y_2 =^? ax_1$$

However the constraints on X are now unsatisfiable: the corresponding recipe can neither start with a constructor nor be an entry of the knowledge base. The constraints in this branch of the case analysis therefore have no solutions.

- ▷ *case 1.1.2: the recipe for Y_2 has a constructor symbol at its root*

The constructor in question is necessarily h . Similarly to case 1 we apply the substitution $\{Y_2 \mapsto h(Y_3)\}$ for some fresh variable $Y_3:2$ which results in the updated constraints:

$$D = X \vdash^? x \wedge Y_3 \vdash^? x \qquad E^2 = Y =^? \langle \text{adec}(ax_2, X), h(Y_3) \rangle \wedge Y_1 =^? \text{adec}(ax_2, X) \wedge Y_2 =^? h(Y_3)$$

Then we observe that X and Y_3 should be unified by uniformity because they deduce the same first-order term x . We have $mgu(X =^? Y_3) = \{Y_3 \mapsto X\}$ (we recall that $\{X \mapsto Y_3\}$ is *not* a valid second-order substitution because Y_3 has a strictly greater type than X) which, after application to the system, results in the updated constraints:

$$D = X \vdash^? x \qquad E^2 = Y =^? \langle \text{adec}(ax_2, X), h(X) \rangle \wedge Y_1 =^? \text{adec}(ax_2, X) \wedge Y_2 =^? h(X) \wedge Y_3 =^? X$$

This will be a typical example of system in solved form. Since we considered all cases and only this branch was successful we conclude that the overall system has a unique mgs which is $mgu(E^2) = \{Y \mapsto \langle \text{adec}(ax_2, X), h(X) \rangle\}$. ◆

Formalisation We will formalise the simplification rules in the next section and focus here on the main three rules (MGS-CONSEQ), (MGS-RES) and (MGS-CONS) mentioned in the above example. For that we reason about a set $R(C^e)$ that represents all recipes that are already used to constraint the solutions of C^e :

$$R(C^e) = st_c(img(mgu(E^2(C^e))), K(C^e) \cup D(C^e)) \cup vars^2(D(C^e))$$

As we saw in the example, the mgs is gradually constructed “within E^2 ”, in the sense that after normalising C^e with the transition system defined in this section, it will have $mgu(E^2)$ as a unique mgs. In particular an invariant of our transition system is that $img(mgu(E^2(C^e)))$ is consequence of $K(C^e)$ and $D(C^e)$, hence the notation $R(C^e)$ is well defined. Formally speaking the transition system relies on three rules of the form

$$C^e \xrightarrow{\Sigma} C^e:\Sigma \quad (\star)$$

for some substitution Σ and under various conditions capturing the possible ways to satisfy the constraints of C^e . The uniformity property is expressed by applying (\star) with

$$\begin{aligned} \Sigma = mgu(\xi =^? \zeta) \text{ for some } \xi \in R(C^e) \cup \mathcal{F}_0, \zeta \in R(C^e), \text{ and provided} & \quad (\text{MGS-CONSEQ}) \\ \Sigma \neq \top, \Sigma \neq \perp, \text{ and } \exists u. (\xi, u), (\zeta, u) \in \text{Conseq}(K(C^e) \cup D(C^e)) & \end{aligned}$$

The result is the unification in C^e of the two second-order terms ξ and ζ that deduce the same term u . It then remains to add rules that express how each term u , $(X \vdash^? u) \in D(C^e)$, can be constructed by the adversary from the knowledge base. When Rule (MGS-CONSEQ) is not applicable we thus apply (\star) under one of the following two conditions. The first one expresses that u is computed by directly using an entry from the knowledge base:

$$\begin{aligned} \Sigma = mgu(X =^? \xi) \neq \perp \text{ where, for some } u \notin \mathcal{X}, \text{ there exist deduction facts} & \quad (\text{MGS-RES}) \\ (X \vdash^? u) \in D(C^e) \text{ and } (\xi \vdash^? v) \in K(C^e) & \end{aligned}$$

Then the last rule expresses that the computation of u starts by applying a constructor f :

$$\begin{aligned} \Sigma = \{X \rightarrow f(X_1, \dots, X_n)\} \text{ where } X_1:k, \dots, X_n:k \text{ are fresh, and there exists} & \quad (\text{MGS-CONS}) \\ \text{a deduction fact } (X:k \vdash^? f(u_1, \dots, u_n)) \in D(C^e) & \end{aligned}$$

As said above we always apply Rule (MGS-CONSEQ) in priority, that is, we add to the last two rules the condition that Rule (MGS-CONSEQ) cannot be applied. This will be crucial in particular when studying the complexity of the procedure in Section 5.2.

4.2.3 First set of simplification rules

To effectively compute most general solutions, the above three rules are applied repeatedly, but some *simplification rules* are also used in between. Their role is to put the constraint systems in a simpler form and in particular to detect the unsatisfiable systems. Other simplification rules,

serving different purposes, will be introduced later in the procedure. The rules here are of two kinds:

1. *simplification rules for formulas* that simply compute mgu and simplify the hypotheses of formulas; and
2. *simplification rules for mgs'* that apply mgu to the rest of the system, and detect unsatisfiability through contradictions or violations of uniformity.

Simplification rules for formulas We first introduce basic simplification rules for formulas that will be used even outside of the computation of most general solutions. We define five sets rules in Figure 7 that apply on constraints of E^1 , E^2 and F .

<i>Misc.</i>	$\neg \top \rightsquigarrow \perp$	$\neg \perp \rightsquigarrow \top$	$\phi \wedge \top \rightsquigarrow \phi$	$\phi \wedge \perp \rightsquigarrow \perp$
<i>Universal vars.</i>	$\forall S \cup \{x\}. H \Leftarrow (x =^? u \wedge \phi) \rightsquigarrow \forall S. H\sigma \Leftarrow \phi\sigma$ if $\sigma = mgu(x =^? u) \neq \perp$ $\forall S \cup \{x\}. H \Leftarrow \phi \rightsquigarrow \forall S. H \Leftarrow \phi$ if $x \notin vars^1(\phi)$			
<i>1st order eq.</i>	$u =^? v \rightsquigarrow mgu(u =^? v)$			
<i>1st order diseq.</i>	$\forall S. \phi \rightsquigarrow \begin{cases} \forall S. \bigvee_{x \in dom(\sigma)} x \neq^? x\sigma & \text{with } \sigma = mgu(\neg\phi) \neq \perp \\ \top & \text{if } mgu(\neg\phi) = \perp \end{cases}$			
<i>2nd order diseq.</i>	$\forall S. \phi \rightsquigarrow \begin{cases} \forall S \cup S'. \bigvee_{X \in dom(\Sigma)} X \neq^? X\Sigma & \text{with } \Sigma = mgu(\neg\phi) \neq \perp \\ & \text{and } S' = vars^2(img(\Sigma)) \setminus vars^2(\phi) \\ \top & \text{if } mgu(\neg\phi) = \perp \end{cases}$			

Figure 7. Simplification rules on formulae

We recall that, in the case of the simplification of second order disequations, the computation of mgu may introduce new variables to match arities (see Section 3.2), hence the need for the universal quantified variables S' . No rules are needed for second-order equations in the context of our decision procedure, since Rules (MGS-CONSEQ), (MGS-RES) and (MGS-CONS) already apply mgu to the entire system. The simplification rules are lifted to extended constraint systems C^e in the natural way, by applying the simplifications to all formulas of $E^1(C^e)$, $E^2(C^e)$ and $F(C^e)$.

Simplification rules for MGS In addition of the rules of Figure 7 we define a couple of other rules specific to the computation of most general solutions. First of all the rule

$$(\Phi, D, E^1 \wedge x =? u, E^2, K, F) \rightsquigarrow (\Phi\sigma, D\sigma, E^1\sigma \wedge x =? u, E^2, K\sigma, F\sigma) \quad (\text{MGS-UNIF})$$

where $x \in \text{vars}^1(E^1, D, \Phi, K, F) \setminus \text{vars}(u)$ and $\sigma = \{x \mapsto u\}$, propagates first-order mgu in the whole system. We also consider the following rule discarding a system with no solutions

$$C^e \rightsquigarrow \perp \quad (\text{MGS-UNSAT})$$

where either of the following three conditions is satisfied:

1. $E^1 = \perp$
2. there exist $\xi, \zeta \in R(C^e)$ such that $(\xi, u), (\zeta, u) \in \text{Conseq}(K(C^e) \cup D(C^e))$ and, writing $\Sigma = \text{mgu}(\xi =? \zeta)$, either $\Sigma = \perp$ or $E^2\Sigma \rightsquigarrow^* \perp$ with the rules of Figure 7
3. there exist $(\forall X:i. X \neq? u) \in D(C^e)$ and $\xi \in \mathcal{T}_i^2$ such that $(\xi, u) \in \text{Conseq}(K(C^e) \cup D(C^e))$

The first condition captures trivially unsatisfiable systems, the second one systems with no uniform solutions, and the third one exhibits a public channel that has been used for an internal communication (which is forbidden by the semantics). Since the whole set of simplification rules (Figure 7 and the above two) is convergent modulo renaming of variables, we denote $C \Downarrow$ a normal form of the extended constraint system C w.r.t. \rightsquigarrow .

4.2.4 Overall procedure and correctness

Description of the procedure The point of the transition systems above is to transform an extended constraint system into a form where it has a unique mgs. More formally:

DEFINITION 4.11 (solved extended constraint system). An extended constraint system C^e is in *solved form* if $C^e \neq \perp$, C^e is irreducible w.r.t. \rightsquigarrow and $\xrightarrow{\Sigma}$, and all deduction facts in $D(C^e)$ have variables as first-order terms.

Intuitively for such constraint systems, $\text{mgu}(E^2(C^e))$ is the unique mgs of C^e . Note however that this method for computing mgs' is only correct under some invariants of our overall procedure. Typically, since second-order equations are not handled by simplification rules, if E^2 contains two contradictory equations $X =? a \wedge X =? b$ for two constants $a \neq b$, our procedure would fail to detect the contradiction. If we define the reduction relation $\xRightarrow{\Sigma} \Downarrow$ as the reflexive transitive closure of the composition of relations $\xrightarrow{\sigma} \Downarrow$, then under the invariants of the procedure we compute a set of most general solutions of C^e as the set $\{\Sigma_{|\text{vars}^2(C^e)} \mid C^e \xRightarrow{\Sigma} \Downarrow C^{e'}, C^{e'} \text{ solved}\}$.

REMARK 4.12 (notation for extended symbolic processes). For convenience we often abuse notations and, if $S = (\mathcal{P}, C, C^e)$ is an extended symbolic process, we write $\text{mgs}(S)$ instead of $\text{mgs}(C^e)$ or say that S is in solved form.

Correctness arguments As mentioned earlier this procedure is only correct under some additional properties verified all along Algorithm 1. For the sake of precision we make explicit mention to these two invariants, $\text{Inv}_{wf}(C^e)$ and $\text{Inv}_{sound}(C^e)$. They are formally defined in Appendix B.1 with a proof that they are preserved during the whole computation of the partition tree, but knowing their exact definition is not necessary to understand the results of this section. The core correctness arguments can be decomposed into following propositions, and are derived from the results proved in Appendix B.4. The first one states that when an extended constraint system cannot be reduced any more then its set of most general solutions is either empty or a singleton:

PROPOSITION 4.13 (mgs of an irreducible system). *Let C^e be an extended constraint system that is irreducible w.r.t. \rightsquigarrow and $\xrightarrow{\Sigma}$, and such that the invariants $\text{Inv}_{wf}(C^e)$ and $\text{Inv}_{sound}(C^e)$ hold. Then*

1. *if C^e is in solved form then $\text{mgs}(C^e) = \{\text{mgu}(E^2(C^e))\}$*
2. *otherwise $\text{mgs}(C^e) = \emptyset$*

The second argument is that applying the mgs constraint-solving rules is correct w.r.t. the solutions of the initial system.

PROPOSITION 4.14 (soundness of one step of the mgs constraint solving). *Let C^e be an extended constraint system with $C^e = C^e \dot{\zeta}$. If $C^e \xrightarrow{\Sigma} \dot{\zeta} C^{e'}$ and $(\Sigma', \sigma) \in \text{Sol}(C^{e'})$ then $(\Sigma'_{|\text{vars}^2(C^e)}, \sigma_{|\text{vars}^1(C^e)}) \in \text{Sol}(C^e)$.*

Finally the last argument formalises than all solutions can be expressed as a sequence of mgs constraint-solving transitions.

PROPOSITION 4.15 (completeness of one step of the mgs constraint solving). *Let C^e be an extended constraint system such that $C^e \dot{\zeta} = C^e$ and the invariants $\text{Inv}_{wf}(C^e)$ and $\text{Inv}_{sound}(C^e)$ hold. We also assume that at least one mgs constraint-solving rule is applicable to C^e . Then for all $(\Sigma, \sigma) \in \text{Sol}(C^e)$, there exist a constraint-solving transition $C^e \xrightarrow{\Sigma_0} \dot{\zeta} C^{e'}$ and $\Sigma \subseteq \Sigma'$, $\sigma \subseteq \sigma'$ such that $(\Sigma', \sigma') \in \text{Sol}(C^{e'})$.*

Together these three results give the partial correctness of the procedure, that is, the correctness of the computation when it terminates. The termination is studied in Section 5.2:

THEOREM 4.16 (partial correctness of mgs computation). *Let C^e be an extended constraint system such that $\text{Inv}_{wf}(C^e)$ and $\text{Inv}_{sound}(C^e)$ hold. Then, assuming there exist no infinite sequences of $\rightarrow \dot{\zeta}$ reductions from C^e , we have*

$$\text{mgs}(C^e) = \{\Sigma_{|\text{vars}^2(C^e)} \mid C^e \dot{\zeta} \xrightarrow{\Sigma} \dot{\zeta} C^{e'}, C^{e'} \text{ solved}\}$$

PROOF. Since a set of mgs' of $C^e \dot{\zeta}$ is also a set of mgs' of C^e , we assume without loss of generality that $C^e \dot{\zeta} = C^e$. Let us write $S = \{\Sigma_{|vars^2(C^e)} \mid C^e \xrightarrow{\Sigma} C^{e'}, C^{e'} \text{ solved}\}$ and prove that S is a set of mgs' of $C^e \dot{\zeta}$. By the termination assumption, we can reason by well-founded induction on the reduction relation $\rightarrow \dot{\zeta}$ from C^e . Using such an induction we can prove the two requirements of the definition, that is:

1. that all $\Sigma \in S$ are solutions of C^e after replacing their second-order variables by fresh constants (base case: Proposition 4.13; inductive case: soundness, i.e., Proposition 4.14).
2. that all solutions of C^e are instances of a substitution of S (base case: Proposition 4.13 again; inductive case: completeness, i.e., Proposition 4.15). ■

4.3 Constraint Solving: symbolic and simplification rules

4.3.1 Symbolic rules

The symbolic rules simply apply the transitions of the symbolic semantics to extended symbolic processes, adding the corresponding constraints to both the symbolic process and the extended constraint system. In that sense most rules are close to identical to those of the symbolic semantics (Section 3.4). Typically the analogue of the rule (s-IN) is:

$$(\{\{u(x).P\}\} \cup \mathcal{P}, C, C^e) \xrightarrow{Y(X)}_s (\{\{P\}\} \cup \mathcal{P}, incr(C), incr(C^e)) \quad (\text{E-IN})$$

where, if $\mathcal{D} \in \{C, C^e\}$, $incr(\mathcal{D}) = \mathcal{D}[D \mapsto D \wedge X \vdash^? x \wedge Y \vdash^? y, E^1 \mapsto E^1 \wedge \sigma]$ with $Y:n$, $X:n$ and y fresh (n size of the domain of the frame of C), and $\sigma \in mgu_{\mathcal{R}}(y =^? u\mu)$, $\mu = mgu(E^1(C)) \neq \perp$. The only rule that is not a trivial extension of the symbolic semantics is the one for outputs that puts a deduction fact in F to model the additional capability this offers to the attacker:

$$(\{\{\bar{u}\langle v \rangle.P\}\} \cup \mathcal{P}, C, C^e) \xrightarrow{\bar{Y}\langle ax_{n+1} \rangle}_s (\{\{P\}\} \cup \mathcal{P}, incr(C), incr(C^e)[F \mapsto F \wedge ax_{n+1} \vdash^? v\sigma\downarrow]) \quad (\text{E-OUT})$$

where, if $\mathcal{D} \in \{C, C^e\}$, $incr(\mathcal{D}) = \mathcal{D}[\Phi \mapsto \Phi \cup \{ax_{n+1} \mapsto v\mu\sigma\downarrow\}, D \mapsto D \wedge Y \vdash^? y, E^1 \mapsto E^1 \wedge \sigma]$ with $Y:n$ and y fresh (n size of the domain of the frame of C), and $\sigma \in mgu_{\mathcal{R}}(y =^? u\mu \wedge v\mu =^? v\mu)$, $\mu = mgu(E^1(C)) \neq \perp$. We omit the definition of the remaining rules corresponding to the other symbolic transitions, all being constructed similarly to (E-IN) by copying the new constraints of C into C^e .

4.3.2 Normalisation rules

We define a new set of simplification rules, called *normalisation rules*, that operate on extended constraint systems. Similarly to the simplification rules for most general solutions introduced in Section 4.2.3 they propagate first-order unifiers across the system and replace unsatisfiable systems by \perp . They also rely on the computation of mgs' of Section 4.2, for example to identify

and remove trivial constraints such as formulas with unsatisfiable hypotheses. They are defined in Figure 8 and commented below (in particular regarding the definition of \simeq_r).

$C^e \rightsquigarrow C^{e'}$	if $C^e \rightsquigarrow C^{e'}$ by rule (MGS-UNIF)	(NORM-UNIF)
$C^e \rightsquigarrow \perp$	if $mgs(C^e) = \emptyset$	(NORM-NO-MGS)
$C^e[E^1 \mapsto E^1 \wedge \forall \tilde{x}.\phi] \rightsquigarrow C^e$	if $mgs(C^e[E^1 \mapsto E^1 \wedge \neg\phi]) = \emptyset$	(NORM-DISEQ)
$C^e[F \mapsto F \wedge \psi] \rightsquigarrow C^e$	if $mgs(C^e[E^1 \mapsto E^1 \wedge \text{hyp}(\psi)]) = \emptyset$	(NORM-FORMULA)
$C^e[F \mapsto F \wedge \psi] \rightsquigarrow C^e$	if $\exists \psi' \in F, \psi' \simeq_r \psi$ and ψ' solved	(NORM-DUPL)

Figure 8. Normalisation rules on extended constraint systems

We recall that we also write $C^e \rightsquigarrow C^{e'}$ if a constraint of $E^1(C^e)$, $E^2(C^e)$ or $F(C^e)$ can be simplified using one of the simplification rules on formulas (Figure 7). The relation \rightsquigarrow can be lifted to sets of (sets of) extended constraint systems or symbolic processes in the natural way. Let us now comment on the rules of Figure 8. Rule (NORM-UNIF) uses the same rule as in the mgs constraint solving to propagate first-order unifiers to the whole system. The next three rules exploit the existence of a most general solution of the constraint system to simplify some constraints:

1. Rule (NORM-NO-MGS) checks whether the constraint system is unsatisfiable, i.e., does not have a most general solution, and in this case transforms it into \perp .
2. Rule (NORM-DISEQ) similarly removes a disequation $\forall \tilde{x}.\phi$ in E^1 when it does not effectively restrict the solutions: for that we require the constraint system not to have solutions that contradict the disequation.
3. Analogously Rule (NORM-FORMULA) removes a formula with unsatisfiable hypotheses. The fact that we only consider the equations among the hypotheses (recall that $\text{hyp}\psi$ omits the hypotheses of ψ that are deduction facts) is due to an invariant of our procedure. We will indeed ensure that formulae are only added to the set F after all deduction facts have been removed from hypotheses by appropriate solving.

Finally Rule (NORM-DUPL) removes an unsolved deduction or equality formula ψ from F when it is subsumed by another formula ψ' . This is formalised by the following notion of equivalence:

DEFINITION 4.17 (head equivalence of formulas). Let $\psi = H \Leftarrow \varphi$ and $\psi' = H' \Leftarrow \varphi'$ be two formulas. We say that ψ and ψ' are *head equivalent*, written $\psi \simeq_r \psi'$, if for some ξ, ζ, u, u' either $H = H' = (\xi \stackrel{?}{=} \zeta)$, or $H = (\xi \vdash^? u)$ and $H' = (\xi \vdash^? u')$.

That is, two formulas are head equivalent if their heads have the same second-order terms (but may differ on their first-order terms), which means they model the same attacker action. In particular if $\psi \simeq_r \psi'$ and ψ' is solved (namely has no hypotheses any more) then the formula ψ is already implied by ψ' which is why Rule (NORM-DUPL) can remove it from F .

4.3.3 Vector-simplification rules

We now define simplification rules that focus on vector, thus called *vector-simplification rules*. They are described in Figure 9 and focus among other things on adding formulas and entries in the knowledge base. This has to be done concurrently on an entire vector component to ensure that the same attacker actions can be performed in all of its elements, that is, that they have statically-equivalent solutions. The rules assume that the constraint systems have been normalised by the normalisation rules (see Figure 8), and one of them uses our custom notation for applying a substitution Σ to a formula (Section 4.2.1, Definition 4.9). Finally, for the sake of succinctness, if $S = (\mathcal{P}, C, C^e)$ is an extended symbolic process we refer as $\Phi(S), E^1(S), E^2(S), \dots$ to the corresponding components of C^e .

Rule (VECT-RM-UNSAT) removes \perp elements from the vector. Rule (VECT-SPLIT) splits a component whenever a common solution would yield statically inequivalent frames. More specifically, the rule separates the constraint systems in Γ^+ in which a given recipe always yields a message (resp. an equality always holds) from the constraint systems in Γ^- in which the same recipe would never yield a message (resp. the same equality would never hold). This is characterised by the fact that a deduction (resp. equality) formula is solved in some constraint systems and not in the others. Rule (VECT-ADD-CONSEQ) adds a solved deduction formula from $F(S)$ to $K(S)$ when this formula is solved in the entire component Γ and the new knowledge-base entries are not redundant with existing ones. Finally, when an equality fact $\xi =_f^? \zeta$ should hold in one constraint system, Rule (VECT-ADD-FORMULA) adds it to the entire component Γ (with appropriate hypotheses). Observe that we use in this rule the placeholder formula $\psi = \forall X, Y, z. X =_f^? Y \Leftarrow (X \vdash^? z \wedge Y \vdash^? z)$, introduced in Section 4.1.1, stating that two recipes deducing the same term should verify an equality fact.

4.4 Constraint Solving: case distinction rules

Our case distinction rules take the form of a transition system on vectors \mathbb{S} of extended symbolic processes similarly to the vector-simplification rules. There are three different rules, each operating in a similar manner: given a vector $\mathbb{S} \cup \{\Gamma\}$, all rules perform a transformation of the following form on one component Γ :

$$\mathbb{S} \cup \{\Gamma\} \rightarrow \mathbb{S} \cup \{\Gamma^+, \Gamma^-\} \quad (\star\star)$$

$$\mathbb{S} \cup \{\Gamma \cup \{(\mathcal{P}, \mathcal{C}, \perp)\}\} \rightsquigarrow \mathbb{S} \cup \{\Gamma\} \quad (\text{VECT-RM-UNSAT})$$

$$\mathbb{S} \cup \{\Gamma\} \rightsquigarrow \mathbb{S} \cup \{\Gamma^+, \Gamma^-\} \quad (\text{VECT-SPLIT})$$

if Γ^+, Γ^- is a partition of Γ and there exists a formula ψ such that

1. $\forall S \in \Gamma^+, \exists \psi' \in F(S), \psi \simeq_r \psi'$ and ψ' solved; and
2. $\forall S \in \Gamma^-, \forall \psi' \in F(S), \psi \not\simeq_r \psi'$

$$\mathbb{S} \cup \{\Gamma\} \rightsquigarrow \mathbb{S} \cup \{\{S[K \mapsto K \wedge \xi \vdash^? u_S] \mid S \in \Gamma\}\} \quad (\text{VECT-ADD-CONSEQ})$$

if for all $S \in \Gamma$, S is solved, $\xi \vdash^? u_S \in F(S)$ and for all second-order term ζ ,

$$(\zeta, u_S) \notin \text{Conseq}(K(S) \cup D(S))$$

$$\mathbb{S} \cup \{\Gamma\} \rightsquigarrow \mathbb{S} \cup \{\{S[F \mapsto F \wedge \psi:(\Sigma, S)] \mid S \in \Gamma\}\} \quad (\text{VECT-ADD-FORMULA})$$

if $\psi = \forall X, Y, z. X \stackrel{?}{=} Y \Leftarrow (X \vdash^? z \wedge Y \vdash^? z)$ and $\Sigma = \{X \mapsto \xi, Y \mapsto \zeta\}$, and for all $S \in \Gamma$,

1. S is solved
2. $F(S)$ contains a formula of the form $\xi \vdash^? u_S$. Besides, there should exist $S \in \Gamma$, such that $(\zeta, u_S) \in \text{Conseq}(K(S) \cup D(S))$.
3. for all $(\zeta_1 \stackrel{?}{=} \zeta_2 \Leftarrow \varphi) \in F(S)$, $\zeta_1 \neq \xi$ and $\zeta_2 \neq \xi$

Figure 9. Vector-simplification rules for sets of sets of extended symbolic processes

where Γ^+ (the *positive branch*) is intuitively obtained by applying a mgs Σ on each symbolic processes of Γ and Γ^- (the *negative branch*) by adding the formula $\neg\Sigma$ to each symbolic process $S \in \Gamma$, where

$$\neg\Sigma = \forall S. \bigvee_{X \in \text{dom}(\Sigma)} X \neq^? X\Sigma \quad \text{with } S = \text{vars}^2(\text{img}(\Sigma)) \setminus \text{vars}^2(\Gamma)$$

Intuitively this refines the component Γ by considering the cases where Σ is a solution or not. After that, normalising the refined components Γ^+ and Γ^- with the simplification rules—in particular Rules (VECT-RM-UNSAT) and (VECT-SPLIT)—will discard impossibles cases and separate processes with newly-found non-statically-equivalent solutions. The three case distinction rules (SAT), (EQ) and (REW) are presented in the next sections by specifying how Γ^+ and Γ^- are computed from Γ . They are applied using a particular strategy defined by the following ordering on rules (where $<$ means “has priority over”):

$$\text{SAT} < \text{EQ} < \text{REW}$$

Note that this ordering is mostly arbitrary: only the minimality of SAT will be needed in Section 5.2 for complexity. The other inequalities are only there to reduce the number of cases to be considered in proofs.

4.4.1 Rule Sat

The first rule focuses on satisfiability: its goal is to separate extended constraint systems of Γ that do not have the same solutions. For example if we have $S = (\mathcal{P}, C, C^e) \in \Gamma$ and $\Sigma \in mgs(C^e)$, all other symbolic processes $S' \in \Gamma$ should also have a solution that is an instance of Σ (and if not, the component Γ should be split to separate S and S'). In particular this ensures that when this rule cannot be applied any more, all extended constraint system in Γ share a common, unique mgs (in particular they are in solved form). The same mechanism can be used to consider the solutions Σ making trivial some disequations of E^1 or hypotheses of some formulas in F . In particular the normalisation rules defined earlier in Section 4.3.2 will then handle the now trivial or unsatisfiable constraints. All this can be formalised as an instance of $(\star\star)$ with:

$$\left| \begin{array}{l} \text{For all } \Sigma, \Gamma, \\ \Gamma^+ = \{S:\Sigma \mid S \in \Gamma\} \\ \Gamma^- = \{S[E^2 \mapsto E^2 \wedge \neg\Sigma] \mid S \in \Gamma\} \end{array} \right. \quad (\text{SAT})$$

where there exists $S \in \Gamma$ such that either

1. S not solved and $\Sigma \in mgs(S)$; or otherwise
2. there exists $\psi \in F(S)$ not solved and $\Sigma \in mgs(S[E^1 \mapsto E^1 \wedge \text{hyp}(\psi)])$; or
3. $E^1(S)$ contains a disequation $\psi = \forall \tilde{x}.\phi$ and $\Sigma \in mgs(S[E^1 \wedge \psi \mapsto E^1]mgu(\neg\phi))$.

4.4.2 Rule Eq

The second case distinction rule focuses on the static equivalence between solutions of extended constraint systems. More specifically, the rule (EQ) checks whether an entry $\xi_1 \vdash^? u_1$ of one knowledge base of Γ can deduce the same term as another recipe ξ_2 consequence of K . The rule is formalised as an instance of $(\star\star)$ with

$$\left| \begin{array}{l} \text{For all } \Sigma, \Sigma_0, \Gamma, \\ \Gamma^+ = \{S:\Sigma[F \mapsto F \wedge \psi:(\Sigma_0\Sigma, S:\Sigma)] \mid S \in \Gamma\} \\ \Gamma^- = \{S[E^2 \mapsto E^2 \wedge \neg\Sigma] \mid S \in \Gamma\} \end{array} \right. \quad (\text{EQ})$$

if there exist $S \in \Gamma$, such that $\Sigma \in mgs(S[E^1 \mapsto E^1 \wedge H_E, D \mapsto D \wedge H_D])$, where H_E and H_D are, respectively, the sets of equations and deduction facts of the hypotheses of $\psi:(\Sigma_0, S)$, and:

1. either $\Sigma_0 = \{X \rightarrow \xi_1, Y \rightarrow \xi_2\}$ for some $(\xi_1 \vdash^? u_1), (\xi_2 \vdash^? u_2) \in K(S)$ and for all $(H \leftarrow \varphi) \in F(S)$, $H \neq (\xi_1 \stackrel{?}{=} \xi_2)$; or

2. $\Sigma_0 = \{X \rightarrow \xi_1, Y \rightarrow f(X_1, \dots, X_n)\}$ for some $(\xi_1 \vdash^? u_1) \in K(S)$ and $f/n \in \mathcal{F}_c$ with $X_1:k, \dots, X_n:k$ fresh and for all $(\zeta_1 \stackrel{?}{=} \zeta_2 \Leftarrow \varphi) \in F(S)$, $\zeta_b = \xi_1$ implies $root(\zeta_{1-b}) \neq f$.
 where $k = |dom(\Phi(S))|$ and $\psi = \forall X, Y, z. X \stackrel{?}{=} Y \Leftarrow (X \vdash^? z \wedge Y \vdash^? z)$ with $X:k, Y:k, z$ fresh variables.

Similarly to Rule (VECT-ADD-FORMULA) the rule uses the generic equality formula ψ and the hypotheses of $\psi:(\Sigma_0, S)$ express that $X\Sigma_0$ and $Y\Sigma_0$ deduce the same term. Since a recipe consequence of K can either be coming from a deduction fact in K or be a recipe with a constructor symbol at its root, we consider the two cases 1 and 2 each with the appropriate instantiation Σ_0 of the placeholders X and Y . The side requirements that head-equivalent formulas should not already be present in $F(S)$ are simply here for termination purpose, thus avoiding infinite aggregation of redundant formulas.

4.4.3 Rule Rew

The third case distinction rule focuses on saturating the knowledge base. For example when outputting a term u , the corresponding symbolic rule (E-OUT) will add a deduction fact $ax_n \vdash^? u$ to F ; the rule (REW) will apply rewrite rules on u to determine whether new messages can be learned by the attacker. Typically if $u = \langle u_1, u_2 \rangle$ the following actions will happen:

1. after $ax_n \vdash^? u$ has been added to F by the symbolic rule (E-OUT), it will be copied to the knowledge base K by the simplification rules (VECT-ADD-CONSEQ) (assuming u is not already deducible from any knowledge base of the component)
2. after that, the case-distinction rule (REW) will add the two deduction facts $fst(ax_n) \vdash^? u_1$ and $snd(ax_n) \vdash^? u_2$ to F , which may in turn be transferred to K as well.

More precisely, given a deduction fact $\xi_0 \vdash^? u_0$, the rule checks whether one may apply a rewrite rule $\ell \rightarrow r$ to u_0 , which may require to first apply a context on u_0 (for example if $u_0 = h(a)$ and $\ell = f(g(h(x)))$). For that we introduce a notion of *skeleton* of ℓ .

DEFINITION 4.18 (rewriting skeleton). Let p be a position of a first-order term ℓ . A *skeleton* for (ℓ, p) is a tuple (ξ, t, D) such that $\xi \in \mathcal{T}(\mathcal{F} \cup \mathcal{F}_0 \cup \mathcal{X}^2)$, $t \in \mathcal{T}(\mathcal{F} \cup \mathcal{F}_0 \cup \mathcal{X}^1)$, D is a set of deduction facts and

$$(root(\xi_{|q}), root(t_{|q})) = \begin{cases} (root(\ell_{|q}), root(\ell_{|q})) & \text{for any strict prefix } q \text{ of } p \\ (X_q, x_q) & \text{for any other position } q \text{ of } \xi \end{cases}$$

where the set of variables X_q (resp. x_q), q a position of ξ that is not strict prefix of p , are fresh pairwise distinct second-order (resp. first-order) variables, and D is the set of all the deduction facts $X_q \vdash^? x_q$. The set of all such skeletons (which, notably, are all identical up to variable renaming but may therefore differ on the second-order-variable types) is written $Skel(\ell, p)$.

For a skeleton $(\xi, t, D) \in Skel(\ell, p)$, the recipe ξ represents the context that the attacker will apply on top of the deduction fact $\xi_0 \vdash^? u_0$ at the position p to obtain the left-hand side ℓ .

The term t represents the corresponding generic term on which the rewrite rule will be applied. Finally D is the set of deduction facts linking the variables of ξ and t .

Consider now a component Γ , a symbolic process $S \in \Gamma$, a deduction fact $(\xi_0 \vdash^? u_0) \in K(S)$, and a context C . The first role of Rule (REW) is to saturate the knowledge base, that is, to deduce the new term $C[u_0]\downarrow$ using $C[\xi_0]$. However after adding the new deduction fact $C[\xi_0] \vdash^? C[u_0]\downarrow$ to $F(S)$, a head-equivalent formula should be added to all other symbolic processes of Γ whenever it is possible, so that the vector-simplification rule (VECT-SPLIT) (which separates processes with non-statically-equivalent solutions) only separates $S' \in \Gamma$ from S if $C[\xi_0]$ yields a valid message in S but not in S' . Yet the behaviour of a destructor symbol may be described by multiple rewrite rules: the rewrite rule used to normalise $C[u_0]$ may therefore not be the same as the one used to normalise the term deduced by $C[\xi_0]$ in S' . Because of this we have to add to $F(S')$ all formulas corresponding to using all possible rewrite rules. For that we consider the following set of generic formulas:

$$\text{RewF}(\xi, \ell \rightarrow r, p) = \left\{ \forall S. \xi \vdash^? r' \Leftarrow (D \wedge \text{mgu}(\ell' =^? t)) \left| \begin{array}{l} \ell' \rightarrow r' \in \mathcal{R} \\ (\xi, t, D) \in \text{Skel}(\ell, p) \\ S = \text{vars}(D, \ell') \end{array} \right. \right\}$$

Let us now give a complete example to illustrate all these notions. The goal is to detail what formulas will be added to F by Rule (REW) on a concrete case as the actual definition of the rule is quite technical and hard to read—although the intuition behind it is rather simple.

EXAMPLE 4.19. Consider a rewriting system defined by a binary symbol h and the two rewrite rules

$$\text{getOther}(h(x, y), x) \rightarrow y \qquad \text{getOther}(h(x, y), y) \rightarrow x$$

These two rewrite rules give access to either argument of h assuming the other one is known. Now consider a component Γ containing two extended symbolic processes S_1 and S_2 with the respective frames, given $k, k', s, s' \in \mathcal{N}$:

$$\Phi(S_1) = \{ax_1 \mapsto h(k, k'), ax_2 \mapsto k\} \qquad \Phi(S_2) = \{ax_1 \mapsto h(s, s'), ax_2 \mapsto s'\}$$

They are statically equivalent, even if the recipe $\text{getOther}(ax_1, ax_2)$ is not normalised using the same rewrite rule in $\Phi(S_1)$ and $\Phi(S_2)$. We assume that $K(S_1) = ax_1 \vdash^? h(k, k') \wedge ax_2 \vdash^? k$ and $K(S_2) = ax_1 \vdash^? h(s, s') \wedge ax_2 \vdash^? s'$. We describe the application of Rule (REW) that uses the rewrite rule $(\ell \rightarrow r) = (\text{getOther}(h(x, y), x) \rightarrow y)$ to deduce a new term in S_1 by putting ax_1 at the position of $h(x, y)$ in ℓ , i.e., position 1. To begin the rule considers a skeleton $(\xi, t, D) \in \text{Skel}(\ell, 1)$:

$$\xi = \text{getOther}(X_1, X_2) \qquad t = \text{getOther}(x_1, x_2) \qquad D = X_1 \vdash^? x_1 \wedge X_2 \vdash^? x_2$$

The set $\text{RewF}(\xi, \ell \rightarrow r, 1)$ therefore contains the following two formulas (normalised by the simplification rules on formulae):

$$\begin{aligned}\psi_1 &= \forall X_1, X_2, x, y. \text{getOther}(X_1, X_2) \vdash^? y \Leftarrow (X_1 \vdash^? h(x, y) \wedge X_2 \vdash^? x) \\ \psi_2 &= \forall X_1, X_2, x, y. \text{getOther}(X_1, X_2) \vdash^? x \Leftarrow (X_1 \vdash^? h(x, y) \wedge X_2 \vdash^? y)\end{aligned}$$

They are only generic formulas indicating that when the left side of a rewrite rule can be computed then the right side can be computed as well. Since our goal is to apply the rewrite rule $\ell \rightarrow r$ where the deduction fact $ax_1 \vdash^? h(k, k')$ is used to deduce the subterm ℓ_1 , we have to replace the variable at position 1 in ξ , namely X_1 , by ax_1 in these formulas. We do this by applying the substitution $\Sigma_0 = \{X_1 \mapsto ax_1\}$ to ψ_1, ψ_2 (in the sense defined in Section 4.2.1, again normalised by simplification rules):

$$\begin{aligned}\psi_1:(\Sigma_0, S_1) &= \forall X_2. \text{getOther}(ax_1, X_2) \vdash^? k' \Leftarrow X_2 \vdash^? k \\ \psi_2:(\Sigma_0, S_1) &= \forall X_2. \text{getOther}(ax_1, X_2) \vdash^? k \Leftarrow X_2 \vdash^? k'\end{aligned}$$

Rule (REW) will then compute most general solutions to instantiate X_2 in a way that satisfies the hypotheses of these formulas. In the case of the first formula we have the unique solution $\text{mgs}(S_1[D \mapsto D \wedge X_2 \vdash^? k]) = \{\Sigma\}$ where $\Sigma = \{X_2 \mapsto ax_2\}$; the algorithm will therefore add the following deduction fact to $F(S_1)$:

$$\psi_1:(\Sigma_0\Sigma, S_1:\Sigma) = \text{getOther}(ax_1, ax_2) \vdash^? k'$$

On the contrary, the algorithm could not have added the second formula: we have $\text{mgs}(S_1[D \mapsto D \wedge X_2 \vdash^? k']) = \emptyset$, meaning that no solutions satisfy its hypotheses. Then we are almost done: as explained earlier it only remains to add a head-equivalent formula to $F(S_2)$, if any, so that the vector-simplification rule (VECT-SPLIT) does not split Γ if the recipe $\text{getOther}(ax_1, ax_2)$ yields a valid message in both S_1 and S_2 . That is, we should add to $F(S_2)$:

$$\begin{aligned}\psi_1:(\Sigma_0\Sigma, S_2:\Sigma) &= (\text{getOther}(ax_1, ax_2) \vdash^? s' \Leftarrow s' =^? s) \\ \psi_2:(\Sigma_0\Sigma, S_2:\Sigma) &= (\text{getOther}(ax_1, ax_2) \vdash^? s \Leftarrow s' =^? s')\end{aligned}$$

This time the situation is reversed compared to S_1 : the first formula has unsatisfiable hypotheses (and will therefore be discarded at the next round of normalisation rules by Rule (NORM-FORMULA)) and the second one will be simplified to $\text{getOther}(ax_1, ax_2) \vdash^? s$. This illustrates why we add one formula for each rewrite rule: should we have only considered ψ_1 , we would have missed the head-equivalent formula $\text{getOther}(ax_1, ax_2) \vdash^? s$ in S_2 , resulting in the incorrect conclusion that $\Phi(S_1) \not\sim \Phi(S_2)$. \blacklozenge

Let us now formalise Rule (REW) in full generality. It can now be defined as an instance of ($\star\star$) under the following conditions:

For all Σ, Γ ,

$$\begin{aligned}\Gamma^+ &= \{S:\Sigma[F \mapsto F \wedge \mathfrak{F}(S)] \mid S \in \Gamma\} \\ \Gamma^- &= \{S[E^2 \mapsto E^2 \wedge \neg\Sigma] \mid S \in \Gamma\}\end{aligned}\tag{REW}$$

if there exist $S \in \Gamma$, $\ell \rightarrow r \in \mathcal{R}$, p position of ℓ , $\xi \in \mathcal{T}(\mathcal{F} \cup \mathcal{X}_{=k}^2)$ with $k = |\text{dom}(\Phi(S))|$, $\psi_0 \in \text{RewF}(\xi, \ell \rightarrow r, p)$, $(\xi_0 \vdash^? u_0) \in \text{K}(S)$, and a function \mathfrak{F} from subsets of Γ to constraints such that the following conditions are met:

1. $p \neq \varepsilon$ and $\ell|_p \notin \mathcal{X}^1$
2. $\Sigma_0 = \{\xi|_p \rightarrow \xi_0\}$ and $\Sigma \in \text{mgs}(S[D \mapsto D \wedge D(\psi_1), E^1 \mapsto E^1 \wedge \text{hyp}(\psi_1)])$ if $\psi_1 = \psi_0:(\Sigma_0, S)$
3. Σ_1 is an injection from $\text{img}(\Sigma) \setminus \text{vars}^2(\psi_1)$ to fresh constants, and for any such injection Σ'_1 , we have $\psi_0:(\Sigma_0\Sigma\Sigma'_1, S:\Sigma) \notin \text{F}(S)$
4. for all $S' \in \Gamma$, $\mathfrak{F}(S') = \{\psi:(\Sigma_0\Sigma\Sigma_1, S':\Sigma) \mid \psi \in \text{RewF}(\xi, \ell \rightarrow r, p)\}$

The conditions are rather technical but simply capture the steps of the example. Commenting the requirements of the rule, Item 1 ensures that the rewriting is not performed at a trivial position. Items 2 and 3 describe the formulas added to each symbolic process of Γ : just as in the example they are obtained by choosing one $S \in \Gamma$, computing a formula ψ_0 (ψ_1 in the example) corresponding to applying one given rewrite rule $\ell \rightarrow r$ in S , replacing the position p of ℓ by an entry of $\text{K}(S)$ by applying Σ_0 and computing a solution Σ of the hypotheses of the resulting formula. Finally, in Item 4, all other symbolic processes $S' \in \Gamma$ receive the formulas of $\mathfrak{F}(S')$, each attempting to apply a rewrite rule to yield a valid message with the same recipe as in ψ_0 .

Note that the computation of the mgs Σ may leave some second-order variables X_1, \dots, X_n unconstrained because they do not need to be instantiated in a particular way to obtain a solution. This is where Item 3 come into play, replacing these pending variables by fresh constants.

4.5 All in all: computing a partition tree

Overall procedure We make reference to the various constraint-solving relations defined in the previous sections using the following notations:

$$\overset{\text{simpl}}{\rightsquigarrow} : \text{simplification rules on formulas} \qquad \overset{\text{norm}}{\rightsquigarrow} : \text{normalisation rules}$$

$$\overset{\text{vect}}{\rightsquigarrow} : \text{vector-simplification rules}$$

$$\overset{\text{SAT}}{\longrightarrow} : \text{(SAT) case-distinction rule} \qquad \overset{\text{EQ}}{\longrightarrow} : \text{(EQ) case-distinction rule}$$

$$\overset{\text{REW}}{\longrightarrow} : \text{(REW) case-distinction rule}$$

All these transition relations are interpreted as binary relations on vectors. We recall that we call a *component* a set Γ of extended symbolic processes and a *vector* a set \mathbb{S} of components, and

that all $(\mathcal{P}, C, C^e) \in \Gamma$ induce a predicate π on second-order solutions of C such that

$$\text{Sol}^\pi(C) = \{(\Sigma_{|\text{vars}^2(C)}, \sigma_{|\text{vars}^1(C)}) \mid (\Sigma, \sigma) \in \text{Sol}(C^e)\}$$

We therefore propose in Algorithm 1 a procedure to compute the partition tree of two bounded plain processes, where the nodes are labelled by components instead of regular partition-tree configurations; in particular the proof of correctness of this algorithm has to justify that the above predicate π can be defined uniformly across the entire nodes of the computed tree.

Correctness arguments To conclude we mention that the core arguments justifying that Algorithm 1 effectively generates a partition tree can be found in Appendix B.5. Technically, most of the theorem statements rely on a collection of invariants, with a proof of their preservation at each step of the procedure (called with “Inv” names such as Inv_{wf} , $\text{Inv}_{\text{sound}}$, ...).

5. Termination and complexity

5.1 Preliminaries

1. In Section 5.2 we prove that Algorithm 1 uses a *finite number of constraint-solving rules* to compute each branch of the partition tree. This proves the all considered security relations (trace equivalence and inclusion, simulation, (bi)similarity) to be decidable.
2. For complexity purposes we then refine this result in Section 5.3: we prove that Algorithm 1 applies at most an *exponential number of rules* and that the nodes of the resulting partition tree have *most general solutions of exponential (DAG) size*.
3. Relying on these bounds, we show that two processes are not equivalent *iff* there exists a *non-equivalence witness of exponential size* (as defined in Section 3.6). This shows the security relations to be decidable in coNEXP time.
4. Finally we show in Section 5.5 that the security relations are *coNEXP hard*. We also provide a complexity analysis in the pure pi-calculus. All in all:

THEOREM 5.1 (complexity of equivalences). *For bounded processes, the problems TRACEEQ, TRACEINCL, SIMULATION, SIMILARITY, and BISIMILARITY are coNEXP complete for constructor-destructor subterm convergent theories. Besides, in the pure pi calculus, TRACEEQ and TRACEINCL are Π_2 complete, and SIMULATION, SIMILARITY and BISIMILARITY are PSPACE complete.*

Notations We also introduce some notations that will be used in most incoming sections. We recall that we study complexity w.r.t. the DAG size of terms (which provides stronger results compared to complexity bounds w.r.t. the tree size of terms); in particular the DAG size of a substitution σ is $|\sigma|_{\text{dag}} = |\text{st}(\text{img}(\sigma))|$, hence the many occurrences of subterm sets below.

```

// Application of simplification rules, as much as possible
Procedure applySimpl( $\mathcal{S}$  : Vector) : Vector =
  if  $\mathcal{S} \xrightarrow{\text{simpl}} \mathcal{S}'$  then return applySimpl( $\mathcal{S}'$ )
  else if  $\mathcal{S} \xrightarrow{\text{norm}} \mathcal{S}'$  then return applySimpl( $\mathcal{S}'$ )
  else if  $\mathcal{S} \xrightarrow{\text{vect}} \mathcal{S}'$  then return applySimpl( $\mathcal{S}'$ )
  else return  $\mathcal{S}$ 

// Application of case-distinction rules, with simplification rules in between
Procedure applyCase( $\mathcal{S}$  : Vector) : Vector =
  if  $\mathcal{S} \xrightarrow{\text{Sat}} \mathcal{S}'$  then return applyCase(applySimpl( $\mathcal{S}'$ ))
  else if  $\mathcal{S} \xrightarrow{\text{Eq}} \mathcal{S}'$  then return applyCase(applySimpl( $\mathcal{S}'$ ))
  else if  $\mathcal{S} \xrightarrow{\text{Rew}} \mathcal{S}'$  then return applyCase(applySimpl( $\mathcal{S}'$ ))
  else return  $\mathcal{S}$ 

// Generates the subtree rooted on a node labelled by the component  $\Gamma$ 
Procedure generateSubtree( $\Gamma$  : Component) : Tree =
   $\Gamma_{\text{in}} \leftarrow \{S' \mid S \xrightarrow{Y(X)}_s S', S \in \Gamma\}$ 
   $\Gamma_{\text{out}} \leftarrow \{S' \mid S \xrightarrow{\bar{Y}(ax)}_s S', S \in \Gamma\}$ 
  if  $\Gamma_{\text{in}} = \Gamma_{\text{out}} = \emptyset$  then
    return a tree reduced to its root, labelled  $\Gamma$ 
  else
     $\mathcal{S} \leftarrow \text{applyCase}(\text{applySimpl}(\{\Gamma_{\text{in}}, \Gamma_{\text{out}}\}))$ 
     $T \leftarrow$  tree with root  $\Gamma$  and children  $\text{generateSubtree}(\Gamma')$  for
      each  $\Gamma' \in \mathcal{S}$ 
    return  $T$ 

// Generates the root and then an entire partition tree of  $P_1$  and  $P_2$ 
Procedure PTree( $P_1, P_2$  : Processes) : Tree =
   $\Gamma_1 \leftarrow \{S \mid P_1 \xrightarrow{\varepsilon}_s S\}$ 
   $\Gamma_2 \leftarrow \{S \mid P_2 \xrightarrow{\varepsilon}_s S\}$ 
   $\{\Gamma\} \leftarrow \text{applySimpl}(\{\Gamma_1 \cup \Gamma_2\})$  // in the root, simplification rules never
    split the vector
  return  $\text{generateSubtree}(\Gamma)$ 

```

Algorithm 1. Computation of the partition tree, with nodes labelled with components

Given an extended constraint system $C^e = (\Phi, D, E^1, E^2, K, F)$ we write

$$\begin{aligned} \mu^1 &= mgu(E^1) && \text{(first-order mgu)} \\ \mu^2 &= mgu(E^2) && \text{(second-order mgu)} \\ T^1 &= st^1(\text{img}(\Phi\mu^1), \text{img}(\mu^1), K\mu^1, D\mu^1) && \text{(first-order terms)} \\ T^2 &= st^2(\text{img}(\mu^2), K, D) && \text{(second-order terms)} \\ R &= st_c(\text{img}(\mu^2), K \cup D) \cup \text{vars}^2(D) && \text{(solution recipes)} \end{aligned}$$

When the extended constraint system is not clear from context we write explicitly $\mu^1(C^e)$, $\mu^2(C^e)$, $T^1(C^e)$,... Intuitively μ^i are the mgu's of the equations of $E[i]$ and we recall in particular that $mgs(C^e) = \{\mu^2\}$ when C^e is solved (Section 4.2.4, Proposition 4.13). The other notations assume $\mu^1 \neq \perp$ (if $\mu^1 = \perp$, C^e will be discarded by the normalisation rule (MGS-UNSAT) anyway). The sets T^1 and T^2 respectively represent the first-order and second-order terms appearing in the system, while $R \subseteq T^2$ models the set of recipes used to build the solution of C^e (i.e., μ^2) from $K \cup D$. We recall that it is the same set as the one used when defining the constraint-solving rules for most general solutions (Section 4.2.2).

REMARK 5.2 (uniformity of second-order terms across components). Due to an invariant of the procedure (Inv_{str} formalised in Appendix B, Section B.1), we know that all extended constraint systems in a component Γ have the same second-order structure. Here this means that $\mu^2(C_1^e) = \mu^2(C_2^e)$ and $T^2(C_1^e) = T^2(C_2^e)$ for any $(\mathcal{P}_1, C_1, C_1^e), (\mathcal{P}_2, C_2, C_2^e) \in \Gamma$. For this reason we may write $\mu^2(\Gamma)$ or $T^2(\Gamma)$ instead of $\mu^2(C^e)$ or $T^2(C^e)$ for some arbitrary $(\mathcal{P}, C, C^e) \in \Gamma$.

5.2 Termination of the constraint solving

5.2.1 Termination of the computation of most general solutions

We first study the termination of the procedure for computing most general solutions provided in Section 4.2. The proof mostly relies on the following measure that characterises the set of first-order terms of an extended constraint system C^e that are not used in $mgs(C^e)$, that is, that are not deduced by any recipe $\xi \in R(C^e)$:

$$\text{unused}^1(C^e) = \{t \in T^1 \mid t \notin \mathcal{X}^1 \wedge \forall \xi \in R(C^e) \setminus \mathcal{X}^2, (\xi, t) \notin \text{Conseq}(K\mu^1 \cup D\mu^1)\}$$

The simplification rules for mgs do not affect this value (except if C^e is replaced by \perp by (MGS-UNSAT)). The application of (MGS-CONSEQ) will ensure that unused^1 is at least non-increasing, while (MGS-RES) and (MGS-CONS) make it strictly decreasing. We summarise this as the following proposition, proved in Appendix C; we recall that, similarly to the correctness arguments in Section 4, the statements makes reference to some procedure invariants formalised in Appendix B.1:

PROPOSITION 5.3 (decrease of unused first-order terms during constraint solving). *Let C^e be an extended constraint system such that $C^e \xrightarrow{\xi} C^e$ and the invariants $\text{Inv}_{wf}(C^e)$ and $\text{Inv}_{sound}(C^e)$ hold. Then let $C^e \xrightarrow{\Sigma} C^{e'} \neq \perp$. If this transition is derived with:*

1. Rule (MGS-CONSEQ): $|\text{unused}^1(C^{e'})| \leq |\text{unused}^1(C^e)|$
2. Rules (MGS-RES) or (MGS-CONS): $|\text{unused}^1(C^{e'})| < |\text{unused}^1(C^e)|$

Using this proposition we can then easily prove the computation of the set of most general solutions to be terminating, and actually to give an upper bound on its cardinality:

THEOREM 5.4 (termination for most general solutions). *There exist no infinite sequences of transitions w.r.t. \rightarrow_{ξ} . Besides if C^e is an extended constraint system such that the invariants $\text{Inv}_{wf}(C^e)$ and $\text{Inv}_{sound}(C^e)$ hold, we have*

$$|\text{mgs}(C^e)| \leq (|\text{K}(C^e)| + 1)^{|\text{unused}^1(C^e)|}$$

PROOF. First of all we observe that consecutive applications of Rule (MGS-CONSEQ) are terminating, since applying this rule strictly decrease the cardinality of the set $m(C^e)$ of parameters (ξ, ζ) the rule can be applied with. Combining this with Proposition 5.3 we obtain that if $C^e \xrightarrow{\Sigma} C^{e'} \neq \perp$ then $C^e < C^{e'}$ w.r.t. the lexicographic composition of unused^1 and m .

Besides consecutive applications of Rule (MGS-CONSEQ) are also confluent by unicity of mgu's. For the same reason the applications of Rules (MGS-RES) or (MGS-CONS) can be performed on one deterministically-chosen deduction fact $(X \vdash^? u) \in \text{D}$. We therefore obtain $\text{mgs}(C^e) = \{\Sigma_{|\text{vars}^2(C^e)} \mid C^e \xrightarrow{\Sigma} C^{e'} \text{ normalised, } C^{e'} \text{ solved}\}$ where a reduction $C^e \xrightarrow{\Sigma} C^{e'}$ is said to be *normalised* when all applications of Rule (MGS-CONSEQ) and the choice of deduction facts in Rules (MGS-RES) or (MGS-CONS) are done in a fixed, deterministic way. Since for any C^e , there are at most $|\text{K}(C^e)|$ normalised applications of Rule (MGS-RES) and 1 normalised application of Rule (MGS-CONS), we deduce by Proposition 5.3 that $|\text{mgs}(C^e)| \leq (|\text{K}(C^e)| + 1)^{|\text{unused}^1(C^e)|}$. ■

5.2.2 Termination of the computation of partition trees

To bound the number of rule applications in Algorithm 1 we define a well-founded measure that decreases after each case-distinction, simplification, normalisation and vector-simplification rules. More precisely the rule applications are always of the form

$$\mathbb{S} \cup \{\Gamma\} \rightarrow \mathbb{S} \cup \{\Gamma_1, \dots, \Gamma_p\} \quad p \in \{1, 2\}$$

and we show that for all $i \in \llbracket 1, p \rrbracket$, $\Gamma > \Gamma_i$ w.r.t. to a well-founded measure on components (under the invariants of the procedure defined in Appendix B). This therefore bounds the number of rule applications to compute a given branch of the partition tree. The measure in question is a tuple of 9 integer components that is ordered w.r.t. the lexicographic ordering.

Measure 1: sizes of the processes As first element of the measure, we compute a maximum on the sizes of the processes in the multisets \mathcal{P} , that is,

$$M_1(\Gamma) = \max_{(\mathcal{P}, C, C^e) \in \Gamma} \sum_{R \in \mathcal{P}} |R|_{\text{dag}} \quad (\text{Meas. 1})$$

Notice that this stays unchanged for any simplification or case distinction rules but strictly decreases when applying the extended symbolic transitions.

Measure 2: Number of constraint systems The third element of the measure considers the number of extended symbolic processes in the set, i.e., $|\Gamma|$, that may increase only when applying a symbolic transition; however it strictly decreases when applying the simplification rules (VECT-SPLIT) and (VECT-RM-UNSAT). Moreover it also strictly decreases for the positive branch of Rule (SAT) when applied with the case 3 of its application conditions. In such a case, we consider a disequation ψ and a mgs Σ of C_j^e that does not satisfy ψ , which will lead to at least one $S \in \Gamma$ being discarded by the simplification rule (VECT-RM-UNSAT).

$$M_2(\Gamma) = |\Gamma| \quad (\text{Meas. 2})$$

Measure 3: Number of terms not consequence Given C^e an extended symbolic constraint system, let us consider the following set representing the set of terms that are not consequence of $K(C^e)$ and $D(C^e)$:

$$\text{set}_K(C^e) = \{t \in T^1 \mid \forall \xi, (\xi, t) \notin \text{Conseq}(K(C^e) \cup D(C^e))\}$$

Typically it corresponds to the terms that are not deducible by the attacker but could potentially be (because the knowledge base is not saturated yet). In fact, when the simplification Rule (VECT-ADD-CONSEQ) is applied, i.e., when a deduction fact $\xi \vdash^? u$ is added to $K(C^e)$, the term u is necessarily a subterm of the frame by the invariant $\text{Inv}_{wf}(C^e)$. Moreover by definition of Rule (VECT-ADD-CONSEQ) we know that u is not already consequence, meaning that the size of $\text{set}_K(C^e)$ will strictly decrease. Finally the case distinction rules never increase the number of elements of $\text{set}_K(C^e)$: indeed they all consist of applying substitutions Σ that are most general solutions of some systems having $K(C^e)$ as their knowledge base, hence their first-order terms are consequence by K-basis. All in all we choose the following component:

$$M_3(\Gamma) = \min_{(\mathcal{P}, C, C^e) \in \Gamma} |\text{set}_K(C^e)| \quad (\text{Meas. 3})$$

Measure 4: Number of unsolved extended constraint systems We recall that the aim of Rule (SAT), case 1 of its application conditions, is to put extended constraint systems in solved form (that is, in a form where they trivially have μ^2 as a unique mgs). If

$$M_4(\Gamma) = |\{(\mathcal{P}, C, C^e) \in \Gamma \mid C^e \text{ unsolved}\}| \quad (\text{Meas. 4})$$

then this measure is strictly decreasing when applying the rule in question. Once a system has a unique mgs, instantiating its second variables does not change this fact and the other case distinction rules are therefore non-increasing w.r.t. this measure.

Measure 5: Applicability of Rule REW The next element represents the number of applications of Rule (REW) that are still possible. Typically, we consider all the parameters of the rule (REW) (the deduction facts from K , the rewrite rule, etc..) on which the rule would be applied with a most general solutions that does not already corresponds to a deduction fact in F . If C^e is an extended constraint system we therefore consider $\text{set}_{\text{REW}}(C^e)$ the set of tuples $(\psi, \ell \rightarrow r, p, \psi_0, \Sigma)$ that satisfy all the application conditions of Rule (REW), and

$$M_5(\Gamma) = \sum_{(\mathcal{P}, C, C^e) \in \Gamma} |\text{set}_{\text{REW}}(C^e)| \quad (\text{Meas. 5})$$

By definition $|\text{set}_{\text{REW}}(C^e)|$ strictly decreases for at least one $(\mathcal{P}, C, C^e) \in \Gamma$ (and non-increasing for the others) when applying Rule (REW). Then let $(\mathcal{P}, C, C^e) \in \Gamma$: the other case distinction rules (SAT) and (EQ) do not increase $|\text{set}_{\text{REW}}(C^e)|$. Indeed if we consider one of their applications $C^e \xrightarrow{\Sigma'} C^e:\Sigma'$, we have

$$\text{set}_{\text{REW}}(C^e:\Sigma') = \{(\psi\Sigma', \ell \rightarrow r, p, \psi_0, \Sigma\Sigma') \mid (\psi, \ell \rightarrow r, p, \psi_0, \Sigma) \in \text{set}_{\text{REW}}(C^e)\}.$$

Note however that $|\text{set}_{\text{REW}}(C^e)|$ may increase by application of Rule (VECT-ADD-CONSEQ) since $|K(C^e)|$ will increase; yet the measure is already decreasing by the component M_3 .

Measure 6: Number of unsolved deduction formulas We recall that Rule (SAT), case 2 of its application conditions, applies a most general solution to remove the hypotheses of one formula $\psi \in F(C^e)$ for some $(\mathcal{P}, C, C^e) \in \Gamma$ (the formula becomes solved in the positive branch, and is removed by (NORM-FORMULA) in the negative branch). This rule application is therefore strictly decreasing w.r.t. the measure

$$M_6(\Gamma) = |\{\psi \in F(C^e) \mid (\mathcal{P}, C, C^e) \in \Gamma, \psi \text{ unsolved deduction formula}\}| \quad (\text{Meas. 6})$$

We only consider deduction formulas ψ for this component. In particular the only rule that may increase this measure (i.e., generate unsolved deduction formulas) is (REW) which is already decreasing w.r.t. the previous component of the measure.

Measure 7: Applicability of Rule EQ Similarly to the analogue component for Rule (REW), we now define the next component that bounds the maximal number of possible applications of Rule (EQ). The application conditions stipulate that it can be applied either

1. on two deduction facts of $K(C_i^e)$, or
2. on one deduction fact of $K(C_i^e)$ in combination with a construction function symbol.

Even if the application conditions also consider a mgs Σ , the number of applications of Rule (EQ) will not depend on their number; this is intuitively because after applying the rule with one arbitrary mgs Σ , the conditions forbid any later applications with identical parameters except Σ . Formally consider for example the case 1 (case (2) follows the same reasoning). The rule is applied on two deduction facts $(\xi_1 \vdash^? u_1), (\xi_2 \vdash^? u_2) \in K(C^e)$. Thus, an equality formula with $\xi_1 \Sigma =_f^? \xi_2 \Sigma$ as head will be added in $F(C^e:\Sigma)$. However, in further applications of the rule, the condition that “for all $(H \Leftarrow \varphi) \in F(C^e:\Sigma)$, $H \neq (\xi_1 =_f^? \xi_2)$ ” will prevent a new application with the same (up to instantiation of Σ) deductions facts from $K(C^e:\Sigma)$.

We therefore conclude that the rule (EQ) can be applied only once per pair of deduction facts in K and once per deduction fact in K and function symbol in \mathcal{F}_c . If C^e is an extended constraint system we therefore consider $\text{set}_{\text{EQ}}(C^e)$ the set of pairs $(\psi, \psi') \in K(C^e)^2$ or $(\psi, f) \in K \times \mathcal{F}_c$ that satisfy all the application conditions of the rule (EQ), and

$$M_7(\Gamma) = \sum_{(\mathcal{P}, C, C^e) \in \Gamma} |\text{set}_{\text{EQ}}(C^e)| \quad (\text{Meas. 7})$$

Measure 8: Number of unsolved equality formulas We now introduce the analogue of Component 7 for equality formulas, that is,

$$M_8(\Gamma) = |\{\psi \in F(C^e) \mid (\mathcal{P}, C, C^e) \in \Gamma, \psi \text{ unsolved equality formula}\}| \quad (\text{Meas. 8})$$

As before Rule (SAT) makes this measure decrease in the case 2 of its application conditions. On the contrary unsolved equality formulas can be generated by two rules: the case distinction rule (EQ) or the vector-simplification rule (VECT-ADD-FORMULA).

Measure 9: Remaining most general solutions So far, every time we showed that one of the previous element of the measure (strictly) decrease by application of a case distinction rule, we always focused on the positive branches of case-distinction rules. The negative branches on the contrary only add recipe disequations to the system, which does not increase any the previous components of the measure *but* strictly decreases the number of most general solutions we can compute for the same instance of the rule. For example, if $\Sigma \in \text{mgs}(C^e)$ then $|\text{mgs}(C^e)| > |\text{mgs}(C^e[E^2 \wedge \neg\Sigma])|$. Hence it suffices to consider the last component:

$$M_9(\Gamma) = \left| \left\{ \Sigma \mid \begin{array}{l} \text{there exists a case distinction rule applicable} \\ \text{from } C^e \text{ with parameter } \Sigma, (\mathcal{P}, C, C^e) \in \Gamma \end{array} \right\} \right| \quad (\text{Meas. 9})$$

Conclusion This gives the termination of the algorithm for computing $T \in \text{PTree}(P, Q)$. We study more precisely the Components 1 to 9 of the measure in Appendix C and prove that they can all be bound by an exponential in $|P, Q, \mathcal{R}|_{\text{dag}}$ (with \mathcal{R} the rewriting system, implicitly including the signature). Hence:

THEOREM 5.5 (termination for partition trees). *For all P, Q plain processes, Algorithm 1 terminates with arguments P, Q . Moreover each branch of the resulting tree is generated by applying at most an exponential number (in $|P, Q, \mathcal{R}|_{\text{dag}}$) of rules, not counting the negative branches of case distinction rules.*

5.3 Bounding the size of most general solutions

5.3.1 Overall approach

Objective We now focus on the theoretical complexity of the decision problems TRACEEQ, BISIMILARITY, SIMULATION, ... Our goal for now is to prove that they are all decidable in coNEXP and the core argument to achieve this is to prove the theorem:

THEOREM 5.6 (size of most general solutions). *If T is a partition tree of P, Q (w.r.t. a rewriting system \mathcal{R}) generated by Algorithm 1, then for all nodes n of T , $|mgs(n)|_{\text{dag}}$ is exponential in $|P, Q, \mathcal{R}|_{\text{dag}}$.*

We will detail in Section 5.4 how to derive a coNEXP decision procedure for equivalence properties by using this result. To bound the size of most general solutions we rely on the results previously established in Section 5.2.1: in the final partition tree $mgs(n) = \mu^2(\Gamma(n))$ and it therefore suffices to prove that for all nodes, $|\mu^2(\Gamma(n))|_{\text{dag}}$ is exponential in $|P, Q, \mathcal{R}|_{\text{dag}}$. However we will instead study the easier-to-track bound:

$$|T^2(\Gamma(n))| \geq |st(\text{img}(\mu^2(\Gamma(n))))| = |\mu^2(\Gamma(n))|_{\text{dag}}$$

Evolution of second-order terms Let us now consider each constraint-solving rule and determine how T^2 evolves along the components along a branch of the partition tree.

1. *Symbolic rules:* only Rules (E-IN) and (E-OUT) increase the size of T^2 by adding at most two new second order variables.
2. *Simplification, normalisation, vector-simplification rules:* only Rule (VECT-ADD-CONSEQ) may increase the size of $T^2(\Gamma)$. Indeed, it transfers a deduction fact from F in K for each extended constraint systems in the current component Γ .
3. *Case distinction rules:* the positive branches of these rules increase the size of T^2 whereas the negative branches leave it unchanged.

It therefore suffices to prove the following result to obtain Theorem 5.6:

PROPOSITION 5.7 (evolution of second-order terms in partition trees). *If $\mathbb{S} \cup \{\Gamma\} \rightarrow \mathbb{S} \cup \mathbb{S}'$ where $\mathbb{S}' = \{\Gamma'\}$ is obtained by Rule (VECT-ADD-CONSEQ) or $\Gamma' \in \mathbb{S}'$ is the positive branch of a case distinction rule, then $|T^2(\Gamma')|_{\text{dag}} - |T^2(\Gamma)|_{\text{dag}}$ is bounded by a polynomial in $|P, Q, \mathcal{R}|_{\text{dag}}$.*

Indeed we recall that by Theorem 5.5, we already know that each branch of the partition tree is obtained after applying at most an exponential number of rules (negative branches of

case distinction rules excluded). Hence we obtain the expected exponential bound on T^2 when combined with the above proposition. The remaining of Section 5.3 is dedicated to its proof.

5.3.2 Bounding the increase of the second-order terms

When applying a mgs We first study the growth of $T^2(C^e)$ when applying a mgs to C^e , which means proving Theorem 5.6 in the case of Rule (SAT). Similarly to our previous results on most general solutions (Section 5.2.1), our bounds depend on $unused^1(C^e)$ the number of first-order terms of C^e that are not already used in the solution, i.e., in μ^2 . We also recall that by Proposition 5.3, this measure is non-increasing when applying any of the mgs simplification and constraint-solving rules, and is even strictly decreasing in the case of Rule (MGS-RES) and (MGS-CONS). Let us now show that its growth is actually inverted compared to T^2 , that is, how much T^2 increases can be bounded by how much $unused^1$ decreases:

PROPOSITION 5.8 (evolution of second-order terms when applying mgs). *For all extended processes C^e that verify the invariants $Inv_{wf}(C^e)$ and $Inv_{sound}(C^e)$, we have*

$$\forall \Sigma \in mgs(C^e), |T^2(C^e:\Sigma)| \leq |T^2(C^e)| + |\mathcal{F}| \times (|unused^1(C^e)| - |unused^1(C^e:\Sigma)|)$$

PROOF. We assume $|\mathcal{F}| > 0$ by convention. It suffices to prove this property when replacing $C^e:\Sigma$ by $C^{e'}$ for $C^e \rightarrow C^{e'} \neq \perp$ obtained by a mgs simplification or constraint-solving rules. We perform a case analysis on the rule in question.

▷ *case 1: simplification rule on formulas*

The simplification rules on formulas only affect first-order terms and second-order disequations and we therefore have $T^2(C^{e'}) = T^2(C^e)$.

▷ *case 2: mgs simplification rule*

We only need to consider Rule (MGS-UNIF). Since it only affects first-order terms, the reasoning is identical to the previous case.

▷ *case 3: mgs constraint-solving rule*

Rules (MGS-CONSEQ) and (MGS-RES) apply a second-order substitution $\Sigma = mgu(\xi =? \zeta)$ to C^e for some $\xi, \zeta \in T^2(C^e)$. In particular we deduce that $|T^2(C^{e'})| \leq |T^2(C^e)|$ and the conclusion thus follows from the fact that $unused^1(C^e) - unused^1(C^{e'}) \geq 0$ by Proposition 5.3. Finally the only rule that increases $T^2(C^e)$ is the last one, (MGS-CONS), that generates n fresh second-order variables for some constructor symbol f/n . In particular $|T^2(C^{e'})| \leq |T^2(C^e)| + n$, hence the result since $unused^1(C^e) - unused^1(C^{e'}) > 0$ by Proposition 5.3. ■

In particular we obtain Proposition 5.7 for Rule (SAT) case 1, provided we manage to prove that $unused^1(C^e)$ is bounded by a polynomial. We explain in Appendix C how to extend the argument to Rules (EQ), (REW) and (VECT-ADD-CONSEQ).

Bound of unused terms To conclude let us establish the polynomial bound on $|\text{unused}^1(C^e)|$. In order to do so we explore the relation between C and C^e in $(\mathcal{P}, C, C^e) \in \Gamma$. Intuitively $\text{unused}^1(C^e)$ always has less elements than $\text{unused}^1(C)$ because

1. the symbolic rules always add the same constraints to C and C^e , ensuring that $\text{unused}^1(C^e)$ increases at most as much as $\text{unused}^1(C)$ by these rules
2. the other rules leave C untouched and do not make $\text{unused}^1(C^e)$ increase.

PROPOSITION 5.9 (approximation of unused terms). For all $(\mathcal{P}, C, C^e) \in \Gamma$,

$$|\text{unused}^1(C^e)| \leq |\Phi(C)\mu^1(C), \mu^1(C)|_{\text{dag}}$$

PROOF. Considering C instead of C^e , we have the trivial approximation

$$|\text{unused}^1(C)| \leq |T^1(C)| = \text{st}(\Phi(C)\mu^1(C), \mu^1(C)) = |\Phi(C)\mu^1(C), \mu^1(C)|_{\text{dag}}.$$

It therefore suffices to prove that $|\text{unused}^1(C^e)| \leq |\text{unused}^1(C)|$. For that we show that the inequality $|\text{unused}^1(C^e)| \leq |\text{unused}^1(C)|$ is preserved when applying any of the constraint-solving rules.

▷ *case 1: symbolic rules*

These rules add the same constraints to C^e and C (up to an additional deduction fact added to $F(C^e)$ in the case of Rule (E-OUT), but this does not affect $\text{unused}^1(C^e)$). In particular since $E^2(C) = K(C) = \emptyset$, if we consider an instance $(\mathcal{P}, C, C^e) \xrightarrow{\alpha}_s (\mathcal{P}', C', C'^e)$ of a symbolic rule we therefore have

$$|\text{unused}^1(C'^e)| - |\text{unused}^1(C^e)| \leq |\text{unused}^1(C')| - |\text{unused}^1(C)|$$

which gives the expected result.

▷ *case 2: simplification, normalisation, vector-simplification rules*

By definition these rules only affect C^e and leave C untouched, hence the conclusion since these rules do not increase $\text{unused}^1(C^e)$.

▷ *case 3: case distinction rules*

Let us consider $\text{CompatSubs}(C^e)$ the set of substitutions Σ such that the notation $C^e:\Sigma$ is well defined, that is, such that

1. if $\text{dom}(\Sigma) \subseteq \text{vars}^2(D(C^e))$
2. for all $X \in \text{dom}(\Sigma)$, there exists t such that $(X\Sigma, t) \in \text{Conseq}(K(C^e) \cup D')$ where $D' = \{Y \vdash^? u \in D(C^e) \mid Y \notin \text{dom}(\Sigma)\} \cup D_{\text{fresh}}$ with

$$D_{\text{fresh}} = \{Y \vdash^? y \mid Y \in \text{vars}^2(\text{img}(\Sigma|_{\text{vars}^2(C^e)})) \setminus \text{vars}^2(C^e), y \text{ fresh}\}$$

This is intuitively the set of substitutions Σ whose image is constructed from $K(C^e)$, up to the new variables of D_{fresh} introduced by Σ . In particular we have for all $\Sigma \in \text{CompatSubs}(C^e)$, $|\text{unused}^1(C^e:\Sigma)| \leq |\text{unused}^1(C^e)|$ (which follows in more details from Proposition B.11 in Appendix B), hence the conclusion. ■

This relation allows to eventually reduce the problem to give a polynomial bound on $\Phi(C)$ and $\mu^1(C)$ which are only affected by symbolic rules (we recall that the other constraint-solving rules do not modify C). All in all this concludes the proof of the expected polynomial bound:

COROLLARY 5.10 (polynomial evolution of second-order terms). *For all extended processes C^e that verify the invariants $\text{Inv}_{wf}(C^e)$ and $\text{Inv}_{sound}(C^e)$, we have*

$$\forall \Sigma \in \text{mgs}(C^e), |T^2(C^e:\Sigma)| \leq |T^2(C^e)| + 9|P, Q, \mathcal{R}|_{\text{dag}}^3$$

PROOF. We agree on the convention that $|P|_{\text{dag}}$, $|Q|_{\text{dag}}$ and $|\mathcal{R}|_{\text{dag}}$ are strictly positive. By Propositions 5.8 and 5.9, it suffices to prove that for all symbolic traces $P \xrightarrow{\text{tr}}_s (\mathcal{P}, C)$, we have that $|\Phi(C)\mu^1(C), \mu^1(C)|_{\text{dag}} \leq 9|P, \mathcal{R}|_{\text{dag}}^2$ (which, as we will see, is a very rough approximation). For that a quick induction on the length of tr allows to construct a set of $|\text{tr}|$ variables $Y = \{y_i\}_{i=1}^{|\text{tr}|}$ and finite set of equations S such that

1. $\mu^1(C) \in \text{mgu}_{\mathcal{R}}(S)$
2. for all $(u =^? v) \in S$, u (resp. v) is either a subterm of a term appearing in P or a variable of Y
3. for all terms $u \in \text{img}(\Phi(C)\mu^1(C))$, there exists u_0 subterm of a term appearing in P such that $u_0\mu^1(C) = u$

The variables of Y model the fresh channel variables introduced when executing (s-IN) or (s-OUT) transitions, and the set of equations S collects the equality tests performed during the trace and how each variable of P is instantiated by E^1 (including by private communications). Independently from this, by induction on a straightforward algorithm to compute mgu modulo theory, we have if \mathcal{R} is constructor-destructor subterm convergent

$$\begin{aligned} |\text{st}(\sigma)| &\leq |\text{st}(S)| + |\mathcal{R}|_{\text{dag}} \times |\{t \in \text{st}(S) \mid \text{root}(t) \in \mathcal{F}_d\}| \\ &\leq 2|\text{st}(S)| \times |\mathcal{R}|_{\text{dag}} \end{aligned} \tag{E}$$

Altogether we therefore obtain

$$\begin{aligned} |\Phi(C)\mu^1(C), \mu^1(C)|_{\text{dag}} &\leq |\text{st}(P)| + 2|\mu^1(C)|_{\text{dag}} && \text{(by 3)} \\ &\leq |P|_{\text{dag}} + 4|\text{st}(S)| \times |\mathcal{R}|_{\text{dag}} && \text{(by 1 and (E))} \\ &\leq |P|_{\text{dag}} + 4(|P|_{\text{dag}} + |\text{tr}|) |\mathcal{R}|_{\text{dag}} && \text{(by 2)} \\ &\leq 9|P, \mathcal{R}|_{\text{dag}}^2 \end{aligned} \quad \blacksquare$$

5.4 Complexity upper bounds for equivalence properties

5.4.1 Complexity of trace equivalence

The goal of this section is to prove the following theorem:

THEOREM 5.11 (complexity of trace equivalence). *TRACEEQ are coNEXP for bounded processes and constructor-destructor subterm convergent theories.*

The proof relies on the following arguments that were developed in previous sections:

1. *charactering trace inclusion with partition trees:* Theorem 3.20
2. *existence of a mgs of exponential size:* Theorem 5.6
3. *soundness and completeness of the symbolic semantics:* see Proposition 3.15

Using these ingredients we prove the core property:

PROPOSITION 5.12 (witness of non-trace equivalence of exponential size). *Let P_1, P_2 be two plain processes w.r.t. a constructor-destructor subterm convergent rewriting system \mathcal{R} . The following points are equivalent:*

1. $P_1 \not\sqsubseteq_t P_2$
2. *there exists a trace $t : P_1 \xRightarrow{\text{tr}} A_1$ such that $|t|_{\text{dag}}$ is exponential in $|P, Q, \mathcal{R}|_{\text{dag}}$ and for all $P_2 \xRightarrow{\text{tr}} A_2, A_1 \not\sqsubseteq A_2$.*

PROOF. The proof of $2 \Rightarrow 1$ is trivial and we therefore focus on $1 \Rightarrow 2$. Let us assume that $P_1 \not\sqsubseteq_t P_2$, and let $T \in \text{PTree}(P_1, P_2)$ the partition tree computed by Algorithm 1. By Theorem 3.20 we obtain a partition-tree trace $P_1 \xRightarrow{T} (\mathcal{P}, C), n$ such that there exist no traces of the form $P_2 \xRightarrow{T} (\mathcal{P}', C'), n$. But by Theorem 5.6 we know that the (DAG) size of $\text{mgs}(n)$ is of exponential in $|P, Q, \mathcal{R}|_{\text{dag}}$, which gives a solution $(\Sigma, \sigma) \in \text{Sol}^{\pi(n)}(C)$ of exponential size as well by definition of a mgs.

Let us then consider the trace $t : P_1 \xRightarrow{\text{tr}\Sigma} (\mathcal{P}\sigma, \Phi(C)\sigma\downarrow)$ (that exists by soundness of the symbolic semantics) and show that it satisfies the conditions of 2. It is indeed of exponential DAG size. Besides assume by contradiction that there exists a trace $P_2 \xRightarrow{\text{tr}\Sigma} (Q, \Psi)$ such that $\Phi(C)\sigma \sim \Psi$. By using the completeness of the symbolic semantics and the properties of the partition tree (Lemma A.1), we would obtain a symbolic process (\mathcal{P}', C') such that $P_2 \xRightarrow{T} (\mathcal{P}', C'), n$, yielding a contradiction. ■

To obtain a decidability result we also use the following result on static equivalence from [2]:

PROPOSITION 5.13 (witness of non-static equivalence of polynomial size). *If two frames Φ and Ψ are not statically equivalent w.r.t. a subterm convergent rewriting system \mathcal{R} , there exist two recipes ξ and ζ such that $|\xi, \zeta|_{\text{dag}}$ is polynomial in $|\Phi, \Psi, \mathcal{R}|_{\text{dag}}$, $\xi\Phi\downarrow = \zeta\Phi\downarrow$ and $\xi\Psi\downarrow \neq \zeta\Psi\downarrow$.*

Wrapping everything together we obtain the following NEXP decision procedure for non-trace equivalence:

1. Given two processes P_1, P_2 , guess an integer $i \in \llbracket 1, 2 \rrbracket$ and a trace $P_1 \xRightarrow{\text{tr}} (\mathcal{P}, \Phi)$ of exponential size. In particular, although $|\text{dom}(\Phi)| \leq |\text{tr}|$, the sizes of the terms in $\text{img}(\Phi)$ may be exponential as well.
2. For each of the exponentially-many traces of the form $t : P_2 \xRightarrow{\text{tr}} (\mathcal{Q}, \Psi)$, guess two recipes ξ_t, ζ_t of exponential size.
3. if for one such trace t we do not have $\xi_t \Phi \downarrow = \zeta_t \Phi \downarrow \Leftrightarrow \xi_t \Psi \downarrow = \zeta_t \Psi \downarrow$, conclude that $P_1 \not\approx_t P_2$.

5.4.2 Complexity of labelled bisimilarity

The goal of this section is to prove the following theorem:

THEOREM 5.14 (complexity of labelled bisimilarity). *BISIMILARITY is CONEXP for bounded processes and constructor-destructor subterm convergent theories.*

Similarly to trace equivalence we build on the results of the previous sections, this time using the characterisation of labelled bisimilarity based on symbolic witnesses (Theorem 3.25). Given a partition tree with most general solutions of exponential size, our goal is therefore to derive from it a symbolic witness of non-equivalence and a solution of this witness (Definition 3.24), both of exponential size as well.

PROPOSITION 5.15 (witness of non-labelled bisimilarity of exponential size). *Let P_1, P_2 be two plain processes w.r.t. a constructor-destructor subterm convergent rewriting system \mathcal{R} . The following points are equivalent:*

1. $P_1 \not\approx_b P_2$
2. *there exists a witness w for (P_1, P_2) such that $|w|_{\text{dag}}$ is exponential in $|P, Q, \mathcal{R}|_{\text{dag}}$.*

PROOF. The proof of $2 \Rightarrow 1$ is trivial and we therefore focus on $1 \Rightarrow 2$. Let us assume that $P_1 \not\approx_b P_2$, and let $T \in \text{PTree}(P_1, P_2)$ the partition tree computed by Algorithm 1. By Theorem 3.25 we obtain a symbolic witness w_s for $(P_0, P_1, \text{root}(T))$ such that $\text{Sol}(w_s) \neq \emptyset$, and it suffices to prove that there exists a solution of w_s of exponential size (where the size of a solution f_{sol} is $\sum_{N \in \text{dom}(f_{\text{sol}})} |f_{\text{sol}}(N)|_{\text{dag}}$). More precisely we construct by induction on w_s a function f mapping the nodes of w_s to second-order substitutions (not necessarily ground) such that:

1. $(f_{\text{mgs}} f) \in \text{Sol}(w_s)$, where $f_{\text{mgs}}(S, n) = \text{mgs}(n)$ and the notation $f = gh$ is defined by $f(N) = g(N)h(N)$ for all nodes N of w_s
2. for all $f_{\text{sol}} \in \text{Sol}(w_s)$, there exists f' such that $f_{\text{sol}} = f_{\text{mgs}} f f'$

In particular since f_{mgs} is of exponential size by Theorem 5.6, it suffices to ensure that f is of exponential size as well.

▷ case 1: w_s is reduced to a leaf N .

Then it suffices to choose $f(N) = id$.

▷ case 2: w_s has a root labelled (S, n) and children N_1, \dots, N_p labelled $(S_1, n'), \dots, (S_p, n')$

Let us write $S = \{A_0, A_1\}$ with, by definition, a symbolic trace $A_0 \xrightarrow{\alpha}_s A'_0$ such that each trace $A_1 \xrightarrow{\bar{\alpha}}_s A'_1$ corresponds to a child $S_i = \{A'_0, A'_1\}$. We apply the induction hypothesis to the children to obtain their respective functions f_1, \dots, f_p . We recall that $Sol(w_s) \neq \emptyset$ by hypothesis and that all solutions f_{sol} verify $f_{sol}(N_1) = \dots = f_{sol}(N_p)$; thus, since by induction hypothesis all solutions of w_s are instances of $f_{mgs} f_i$, we obtain:

$$mgu(mgs(n')f_1(N_1)q_1 \wedge \dots \wedge mgs(n')f_p(N_p)q_p) \neq \perp$$

for q_1, \dots, q_p fresh variables renamings of $img(f_1(N_1)), \dots, img(f_p(N_p))$, respectively. In particular, assuming without loss of generality that all the $f_i(N_i)$ have the same domain $(vars^2(n') \setminus dom(mgs(n'))) \cup img(mgs(n'))$, we can write

$$\Sigma = mgu(f_1(N_1)q_1 \wedge \dots \wedge f_p(N_p)q_p) \neq \perp$$

Note that this mgu is only polynomially bigger than each $f_i(N_i)$. Since $mgs(n')$ is an instance of $mgs(n)$, we also let Σ_0 such that $mgs(n') = mgs(n)\Sigma_0$. We then conclude the proof by defining f as follows:

1. $f(root(w_s)) = (\Sigma_0\Sigma)_{|vars^2(n)}$
2. for all $i \in \llbracket 1, p \rrbracket$, for all nodes N in the subtree of w_s rooted in N_i , $f(N) = f_i\Sigma$. ■

5.5 Complexity lower bounds

We prove in this section the complexity lower bounds stated in Theorem 5.1.

5.5.1 Extensions of the calculus

We first introduce useful syntax extensions that can be encoded in the original calculus. We point out that that using these encodings does not affect the complexity of deciding the related decision problems, since they rely on polynomial-size encodings.

Internal non-deterministic choice A first classical operator is the *non-deterministic choice*: $P+Q$ is a process that can be executed either as P or as Q . Its operational semantics can therefore be described by adding the following rule to those of Figure 1:

$$(\{\{P+Q\}\} \cup \mathcal{P}, \Phi) \xrightarrow{\star} (\{\{R\}\} \cup \mathcal{P}, \Phi) \quad \text{if } R \in \{P, Q\} \quad \text{(CHOICE)}$$

This reduction can easily be encoded as an internal communication on a fresh private channel. We formalise it by a process transformation $\llbracket \cdot \rrbracket$:

$$\llbracket P + Q \rrbracket \triangleq \bar{s}\langle s \rangle \mid s(x). \llbracket P \rrbracket \mid s(y). \llbracket Q \rrbracket \quad \text{where } s \in \mathcal{N} \text{ and } x, y \in \mathcal{X}^1 \text{ are fresh} \quad (1)$$

and all other cases of the syntax are handled as homomorphic extensions of $\llbracket \cdot \rrbracket$. As for the parallel operator we will sometimes use the big operator \sum assuming right-associativity. The correctness of this translation with respect to \approx_t and \approx_b will be stated later on in this section.

We also introduce the $\text{Choose}(x)$ construct which non-deterministically assigns either 0 or 1 to x . $\text{Choose}(x).P$ silently reduces to either $P\{x \mapsto 0\}$ or $P\{x \mapsto 1\}$ and $\text{Choose}(\vec{x}).P$ is defined as $\text{Choose}(x_1).\text{Choose}(x_2) \dots \text{Choose}(x_n).P$ where $\vec{x} = x_1, \dots, x_n$. Formally, we extend the operational semantics with the rule

$$(\mathcal{P} \cup \{\text{Choose}(x).P\}, \Phi) \xrightarrow{\varepsilon} (\mathcal{P} \cup \{P\{x \mapsto 0\}\}, \Phi) \quad (\text{CHOOSE-0})$$

$$(\mathcal{P} \cup \{\text{Choose}(x).P\}, \Phi) \xrightarrow{\varepsilon} (\mathcal{P} \cup \{P\{x \mapsto 1\}\}, \Phi) \quad (\text{CHOOSE-1})$$

and define

$$\llbracket \text{Choose}(y).P \rrbracket \triangleq (\bar{d}\langle 0 \rangle + \bar{d}\langle 1 \rangle) \mid d(y). \llbracket P \rrbracket \quad \text{with } d \in \mathcal{N} \text{ is fresh}$$

Boolean circuits and formulae Complete problems in complexity theory often involve boolean formulae (e.g., SAT or QBF). The ability to evaluate boolean formulae, or boolean circuits in general, within the applied π -calculus is therefore crucial. We can implement such a feature by the means of private channels and internal communication: each edge of a boolean circuit Γ indeed mimics a channel transmitting a boolean over a network (Figure 10).

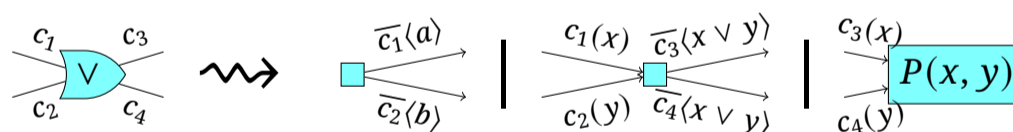


Figure 10. Simulation of an OR-gate within the applied π -calculus

The essence of circuits lies in so-called *logical gates* which are boolean functions with at most two inputs. We consider the fan-out of gates to be possibly more than one in order to model the fact that wires of a gate can be split and connected to the input of different gates. Formally, we assume without loss of generality that the gate has at most two (identical) outputs, to be given as input to other gates. Logical gates usually range over the constants 0 and 1 and the predicates \wedge , \vee and \neg with the usual truth tables but we may use other common operators such as $=$. From that a *boolean circuit* is an acyclic graph of logical gates: each input (resp.

output) of a gate is either isolated or connected to a unique output (resp. input) of another gate, which defines the edges of this graph.

Such a circuit Γ with m isolated inputs and n isolated outputs thus models a boolean function $\Gamma : \mathbb{B}^m \rightarrow \mathbb{B}^n$ (where $\mathbb{B} = \{0, 1\}$). We write $(c_1, c_2, g, c_3, c_4) \in \Gamma$ to state that $g : \mathbb{B}^2 \rightarrow \mathbb{B}$ is a gate of Γ whose inputs are passed through edges c_1 and c_2 and whose output is sent to edges c_3 and c_4 . This notation is naturally lifted to other in-outdegrees.

Embedding into the calculus The syntax of plain processes is now extended with the construction $x_1, \dots, x_n \leftarrow \Gamma(b_1, \dots, b_m).P$ where $\Gamma : \mathbb{B}^m \rightarrow \mathbb{B}^n$ is a circuit, x_1, \dots, x_n variables and b_1, \dots, b_m terms. We fix two distinct terms $0, 1 \in \mathcal{F}_0$ to model \mathbb{B} within the calculus, and the labelled operational semantics is extended with the rule:

$$(\mathcal{P} \cup \{\vec{x} \leftarrow \Gamma(\vec{b}).P\}, \Phi) \xrightarrow{\varepsilon} (\mathcal{P} \cup \{P\{\vec{x} \mapsto \Gamma(\vec{b}\downarrow)\}\}, \Phi) \quad \text{if msg}(\vec{b}) \text{ and } \vec{b}\downarrow \subseteq \mathbb{B} \quad (\text{VALUATE})$$

Now we have to extend the definition of $\llbracket \cdot \rrbracket$ (previous subsection) to handle the new operator. For simplicity we only consider the case where gates have two inputs and two outputs: handling lower arities is straightforward. If $(c_1, c_2, g, c_3, c_4) \in \Gamma$, we first define:

$$\llbracket c_1, c_2, g, c_3, c_4 \rrbracket \triangleq c_1(x).c_2(y). \prod_{b, b' \in \mathbb{B}} \text{if } x = b \text{ then if } y = b' \text{ then } (\overline{c_3}\langle g(b, b') \rangle \mid \overline{c_4}\langle g(b, b') \rangle)$$

where $c_1, c_2, c_3, c_4 \in \mathcal{N}$ (assuming that different circuits in a process do not share edges). To sum it up, we simply see circuit edges as private channels and simulate the logical flow of the gate. It is then easily extended:

$$\llbracket \vec{x} \leftarrow \Gamma(\vec{b}).P \rrbracket \triangleq \left(\prod_{k=1}^m \overline{c_{i_k}}\langle b_k \rangle \right) \mid \left(\prod_{(c_1, c_2, g, c_3, c_4) \in \Gamma} \llbracket c_1, c_2, g, c_3, c_4 \rrbracket \right) \mid c_{o_1}(x_1) \dots c_{o_n}(x_n). \llbracket P \rrbracket$$

where $(c_{i_k})_{k=1}^m$ (resp. $(c_{o_k})_{k=1}^n$) are the isolated input (resp. output) edges of Γ . Note that when b and b' are fixed booleans, $g(b, b')$ denotes the boolean obtained from the truth table of g : we emphasise that g is *not* a function symbol of the signature \mathcal{F} .

REMARK 5.16 (simplifying assumption). We assume that every input of a circuit goes through at least one gate and every circuit has at least one output. This is to avoid irrelevant side cases in proofs.

Correctness of the translation Now we dispose of an extended syntax and semantics as well as a mapping $\llbracket \cdot \rrbracket$ removing the new constructors from a process. The correctness of this translation is proven in Appendix C:

PROPOSITION 5.17 (correctness of the encodings). *Let \approx_t^+ and \approx_b^+ be the notions of trace equivalence and labelled bisimilarity over the extended calculus (the flag $^+$ being omitted outside of*

this lemma). For all extended processes $A = (\mathcal{P}, \Phi)$, the translation $\llbracket A \rrbracket = (\llbracket \mathcal{P} \rrbracket, \Phi) = (\{\llbracket P \rrbracket \mid P \in \mathcal{P}\}, \Phi)$ can be computed in polynomial time, $A \approx_t^+ \llbracket A \rrbracket$ and $A \approx_b^+ \llbracket A \rrbracket$.

REMARK 5.18 (stability of common fragments). As the finite and pure fragments of the applied π -calculus are closed under $\llbracket \cdot \rrbracket$, sums and circuits can be safely used within any intersection of such fragments. The encoding does not use else branches either.

5.5.2 Lower bounds in the pure fragment

We now use the above tool to state our reductions, first, in the pure pi calculus.

Trace equivalence To show that trace equivalence is Π_2 -hard we proceed by a reduction from QBF_2 , that is, the problem of deciding, given φ a boolean formula whose variables are partitioned into $\{\vec{x}\} \cup \{\vec{y}\}$, whether $\forall \vec{x}. \exists \vec{y}. \varphi(\vec{x}, \vec{y}) = 1$. Our goal is to thus to construct two processes A and B such that:

$$A \approx_t B \quad \text{iff} \quad \forall \vec{x}. \exists \vec{y}. \varphi(\vec{x}, \vec{y}) = 0 \quad (2)$$

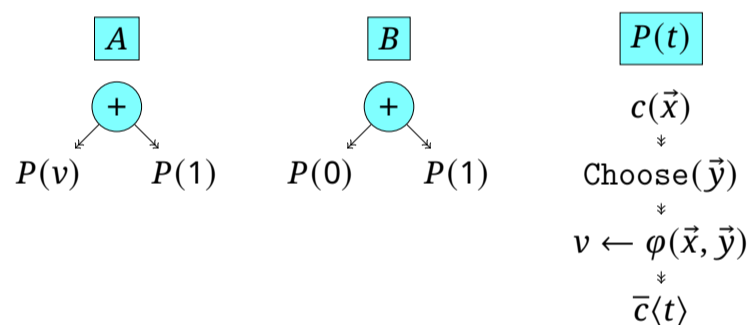


Figure 11. Schematic definition of A and B

Consider three distinct names $c, 0, 1 \in \mathcal{F}_0$ (the last two modelling booleans for the syntax extension of circuit evaluation, see Section 5.5.1). Processes A and B are depicted in Figure 11. Intuitively, the process $P(t)$ (where t is a term which may depend on the variables bound by P) gets a valuation of \vec{x} from the attacker, internally chooses a valuation of \vec{y} , computes the value of $\varphi(\vec{x}, \vec{y})$ using rule VALUATE, and outputs t . From that it is quite easy to see that A and B have the same set of traces iff for all valuation of \vec{x} , there exists a valuation of \vec{y} such that $\varphi(\vec{x}, \vec{y}) = 0$. This reduction is formalised and proved in Appendix D.3.

THEOREM 5.19. *In the pure pi-calculus, TRACEEQ and TRACEINCL are Π_2 -hard for bounded positive processes.*

Note that the hardness for TRACEINCL is directly implied by the hardness of TRACEEQ. This is evidenced by the reduction that, for all processes P, Q , $P \sqsubseteq_t Q$ iff $P + Q \approx_t Q$.

Simulations We now prove that labelled bisimilarity is PSPACE-hard for the positive pure pi calculus by reduction from QBF. This is more involved as QBF allows arbitrary quantifier alternation. Let φ be a boolean formula whose variables are partitioned into $\{x_1, \dots, x_n\} \cup \{y_1, \dots, y_n\}$ for some $n \in \mathbb{N}$. We construct (in polynomial time in the size of φ and n) two processes A and B such that:

$$A \approx_b B \quad \text{iff} \quad A \approx_s B \quad \text{iff} \quad \forall x_1 \exists y_1 \dots \forall x_n \exists y_n. \varphi(x_1, \dots, x_n, y_1, \dots, y_n) = 0 \quad (3)$$

Both QBF and labelled bisimilarity may be seen as bisimulation games: an attacker plays the \exists -quantifiers (selects a transition in a process) whereas a defender responds with the \forall -quantifiers (tries to find a similarly-labelled sequence of transitions in the other process). The role of A and B is to implement this intuitive connection: the attacker moves will be simulated by public inputs $c(x_i)$ and the defender responses by instructions $\text{Choose}(z_i).c(y_i)$. The structure of A and B is then designed to constrain the moves of the two players so that the winning condition of the attacker is exactly $\exists x_1 \forall y_1 \dots \exists x_n \forall y_n. \varphi(\vec{x}, \vec{y}) = 1$.

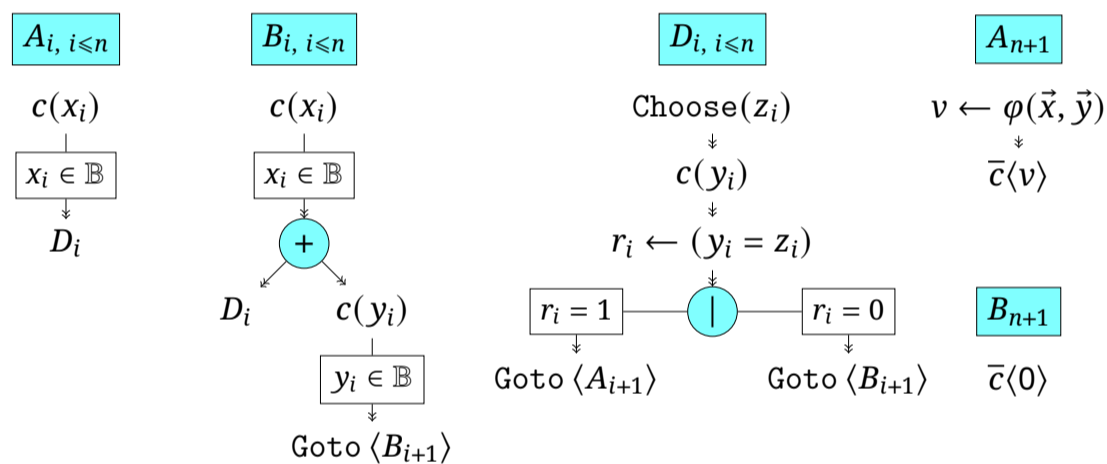


Figure 12. Schematic definition of A_i and B_i

A and B are defined inductively by processes A_i , B_i and D_i , depicted in Figure 12, structured in a way that, in a bisimulation game:

1. the attacker chooses the instance of x_i ;
2. the defender chooses the instance of z_i and can force the attacker to instantiate y_i with the same value (the attacker not doing so allows for a trivial victory of the defender).

The intermediary processes $\text{Goto } \langle A_i \rangle$ and $\text{Goto } \langle B_i \rangle$ intuitively formalise value passing from one index to another, in order to avoid an exponential blowup when encoding n nested tests. Their precise definition and the correctness of the reduction is formalised in Appendix D.3. As before, the hardness of the pre-order follows from the hardness of its symmetric closure.

THEOREM 5.20. *In the pure pi-calculus, SIMULATION, SIMILARITY and BISIMILARITY are PSPACE-hard for bounded positive processes.*

5.5.3 Reduction of SuccinctSAT to process equivalence

We now show that, when cryptographic primitives are modelled by a destructor subterm convergent rewrite system, TRACEEQ, TRACEINCL, SIMULATION, SIMILARITY and BISIMILARITY are coNEXP hard by reducing SuccinctSAT to process equivalence. Consider an instance of SUCCINCTSAT, Γ , with $m+2$ inputs and $n+1$ outputs and we design \mathcal{F} , \mathcal{R} subterm destructor and A and B positive processes such that, for any equivalence relation $\approx \in \{\approx_s, \approx_t, \approx_b\}$, $A \not\approx B$ iff $\llbracket \Gamma \rrbracket_\varphi$ is satisfiable.

Term algebra Terms are built over the following signature:

$\mathcal{F} \triangleq 0, 1,$	(booleans \mathbb{B})
$\text{Node}/2, \pi/2,$	(binary trees)
$h/2,$	(one-way binary hash)
$h_N/2, h_B/2, \text{Test}_N/1, \text{Test}_B/1$	(testable binary hashes)

We equip this term algebra with the rewriting system E containing the following rules modelling subtree extraction (for binary trees) and argument testing (for hashes):

$$\begin{array}{lll} \pi(\text{Node}(x, y), 0) \rightarrow x & \pi(\text{Node}(x, y), 1) \rightarrow y & \\ \text{Test}_N(h_N(\text{Node}(x, y), z)) \rightarrow 1 & \text{Test}_B(h_B(0, z)) \rightarrow 1 & \text{Test}_B(h_B(1, z)) \rightarrow 1 \end{array}$$

In particular \mathcal{R} is subterm and destructor, the destructor symbols being π , Test_N and Test_B . We will also use a shortcut for recursive subtree extraction: if ℓ is a finite sequence of first-order terms, the notation $t|_\ell$ is inductively defined by:

$$t|_\varepsilon \triangleq t \qquad t|_{b.\ell} \triangleq \pi(t, b)|_\ell$$

Core of the reduction Let us give the intuition behind the construction before diving into the formalism. Recall that we are studying a formula in CNF $\llbracket \Gamma \rrbracket_\varphi$ with 2^n variables and 2^m clauses. In particular, given a valuation of its 2^n variables, we can verify in non-deterministic polynomial time in n, m that it falsifies $\llbracket \Gamma \rrbracket_\varphi$:

1. guess an integer $i \in \llbracket 0, 2^m - 1 \rrbracket$ as a sequence of m bits;
2. obtain the three literals of the i^{th} clause of $\llbracket \Gamma \rrbracket_\varphi$ (requiring three runs of the circuit Γ) and verify that the valuation falsifies the disjunction of the three literals.

This non-deterministic verification is the essence our reduction. In the actual processes:

1. a process $\text{CheckTree}(x)$ checks that x is a correct encoding of a valuation, that is, that x is a complete binary tree of height n whose leaves are booleans;
2. a process $\text{CheckSat}(x)$ implements the points 1. and 2. above.

All of this is then formulated as equivalence properties within A and B (see the intermediary lemmas in the next paragraph for details). Intuitively, we want to express the following

statement by equivalence properties: “for all term x , either x is not an encoding of a valuation or falsifies a clause of $\llbracket \Gamma \rrbracket_\phi$ ”. A schematised definition is proposed in Figure 13.

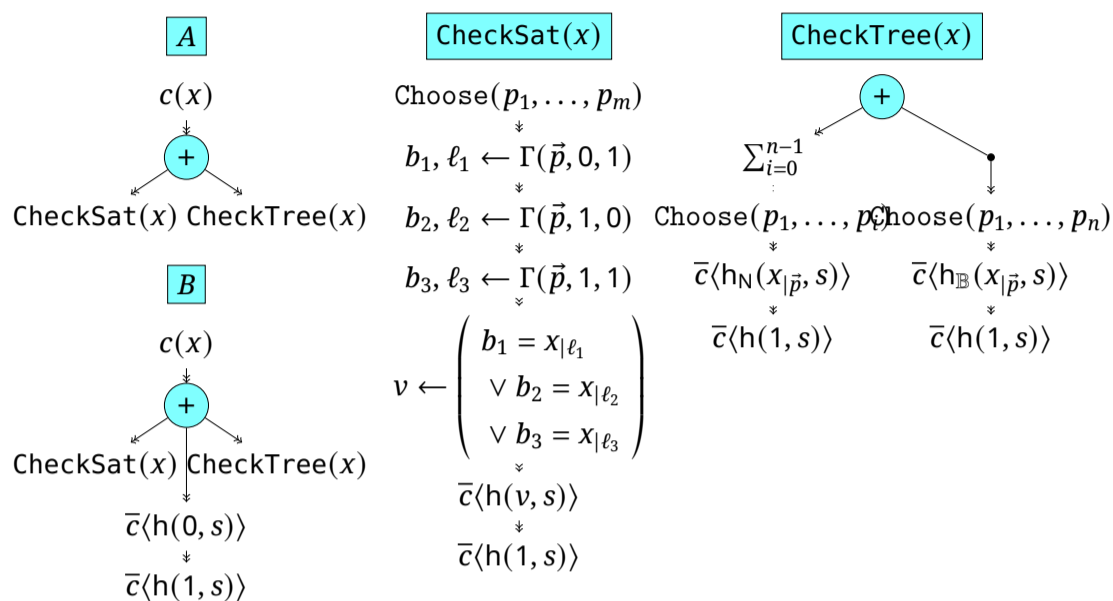


Figure 13. Informal definition of A and B

Formal construction Let us now define the processes depicted in Figure 13 properly; note that all the proofs about the correctness of this construction are relegated to Appendix C but we still state several intermediary lemmas in order to highlight the proof structure. But first of all, let us give a name to a frame which is at the core of our reduction:

$$\Phi_0 = \{ax_1 \mapsto h(0, s), ax_2 \mapsto h(1, s)\}$$

Φ_0 is reached after executing the central branch of B and everything is about knowing under which conditions a frame statically equivalent to Φ_0 can be reached in A . Let us define the processes themselves now. We fix $s \in \mathcal{N}$ and define, if x is a protocol term:

$$\begin{aligned} \text{CheckTree}(x) \triangleq & \sum_{i=0}^{n-1} \left(\text{Choose}(p_1, \dots, p_i). \bar{c}\langle h_N(x|_{p_1 \dots p_i}, s) \rangle. \bar{c}\langle h(1, s) \rangle \right) \\ & + \text{Choose}(p_1, \dots, p_n). \bar{c}\langle h_B(x|_{p_1 \dots p_n}, s) \rangle. \bar{c}\langle h(1, s) \rangle \end{aligned}$$

PROPOSITION 5.21 (correctness of the tree checker). *Let x be a message which is not a complete binary tree of height n with boolean leaves. Then there exists a reduction $\text{CheckTree}(x) \stackrel{\varepsilon}{\Rightarrow} (\llbracket P \rrbracket, \emptyset)$ such that $P \approx_b \bar{c}\langle h(0, s) \rangle. \bar{c}\langle h(1, s) \rangle$.*

Now let us move on to $\text{CheckSat}(x)$. This process binds a lot of variables:

1. $\vec{p} = p_1, \dots, p_m$ models the non-deterministic choice of a clause number in $\llbracket 0, 2^m - 1 \rrbracket$;
2. $b_i, \ell_i, i \in \llbracket 1, 3 \rrbracket$, where ℓ_i is a sequence of n variables, model the literals of the clause chosen above (b_i is the negation bit and ℓ_i the identifier of the variable);

3. v stores whether the chosen clause is satisfied by the valuation modelled by x .

$$\begin{aligned} \text{CheckSat}(x) &\triangleq \text{Choose}(\vec{p}). \\ b_1, \ell_1 &\leftarrow \Gamma(\vec{p}, 0, 1). \\ b_2, \ell_2 &\leftarrow \Gamma(\vec{p}, 1, 0). \\ b_3, \ell_3 &\leftarrow \Gamma(\vec{p}, 1, 1). \\ v &\leftarrow (b_1 = x_{|\ell_1} \vee b_2 = x_{|\ell_2} \vee b_3 = x_{|\ell_3}). \\ &\bar{c}\langle h(v, s) \rangle. \bar{c}\langle h(1, s) \rangle \end{aligned}$$

PROPOSITION 5.22 (correctness of the sat checker). *Let x be a complete binary tree of height n whose leaves are booleans, and val_x be the valuation mapping the variable number i of $[\Gamma]_\varphi$ to $x_{|p_1 \dots p_n} \in \mathbb{B}$ where $p_1 \dots p_n$ is the binary representation of i (i.e., $i = \sum_{k=1}^n p_k 2^{k-1}$). If val_x does not satisfy $[\Gamma]_\varphi$ then there exists $\text{CheckSat}(x) \xrightarrow{\varepsilon} P$ such that $P \approx_b \bar{c}\langle h(0, s) \rangle. \bar{c}\langle h(1, s) \rangle$.*

We can finally wrap up everything by defining A and B and stating the last part of the correctness theorem. We recall that all the proofs can be found in Appendix C.

$$\begin{aligned} A &\triangleq c(x).(\text{CheckSat}(x) + \text{CheckTree}(x)) \\ B &\triangleq c(x).(\text{CheckSat}(x) + \text{CheckTree}(x) + \bar{c}\langle h(0, s) \rangle. \bar{c}\langle h(1, s) \rangle) \end{aligned}$$

PROPOSITION 5.23 (correctness of the reduction). *For any equivalence relation $\approx \in \{\approx_s, \approx_t, \approx_b\}$, $[\Gamma]_\varphi$ is satisfiable iff $A \not\approx B$.*

As a conclusion we obtain the coNEXP hardness of equivalence properties (and their respective pre-orders as a consequence) for constructor-destructor subterm convergent theories. This is stated by the theorem below, which additionally puts an emphasis on the fact that the rewriting system used in our reduction is constant, that is, it does not depend on Γ .

THEOREM 5.24 (hardness of equivalences). *There exists a fixed constructor-destructor subterm convergent rewriting system \mathcal{R} such that the decision problems $\mathcal{R}\text{-TRACEEQ}$, $\mathcal{R}\text{-TRACEINCL}$, $\mathcal{R}\text{-SIMULATION}$, $\mathcal{R}\text{-SIMILARITY}$ and $\mathcal{R}\text{-BISIMILARITY}$ are coNEXP hard for bounded positive processes.*

6. Conclusion and future work

In this article we have studied automated verification of equivalence properties, encompassing both theoretical and practical aspects. We provide tight complexity results for static equivalence, trace equivalence and labelled (bi)similarity (as well as their respective pre-orders), summarised in Table 1. In particular we show that deciding trace equivalence and labelled (bi)similarity for a bounded number of sessions is coNEXP complete for subterm convergent destructor rewrite systems. Finally, we implement the procedure for deciding trace equivalence in the DEEPSEC prototype. As demonstrated through an extensive benchmark (Table 2), our tool is broad in scope and efficient compared to other tools.

Our work opens several directions for future work. It would be interesting to lift the restriction of subterm convergent equational theories to allow for more cryptographic primitives. Similarly, we plan to avoid the restriction to destructor rewrite systems to more general ones. Also, in recent work [30] it was shown that labelled similarity is the same relation as *may testing equivalence* in presence of a probabilistic adversary which motivates the extension of our implementation beyond trace equivalence. The presented procedure for (bi)similarity is however highly non-deterministic and a naive implementation would certainly be inefficient.

Finally, we also plan to extend the DEEPSEC tool with support for other types of properties. The extension provided in this article to simulation and other security relations shows the modularity of our core proof technique, the partition tree, for analysing security properties. For example, a simplified version of the tree could be used to verify more classical (and simpler) *trace properties*, which would significantly rise the scope and usability of DEEPSEC. More generally, since navigation within the tree already allows to verify the complex notion of bisimilarity, we expect that the technique should scale to *hyperproperties* in general. There are few formalisms and results for such properties in the context of security protocols, but hyperlogics fitting our symbolic model have recently been introduced [11]. They allow for example to model fine variants of equivalence relations to capture subtle hypotheses, and their combination to liveness or real-time properties. We expect that our proof techniques would allow to study the decidability and complexity of a large fragment of such logics.

References

- [1] Martín Abadi, Bruno Blanchet, and Cédric Fournet. The applied pi calculus: mobile values, new names, and secure communication. *J. ACM*, 65(1):1:1–1:41, 2018. [DOI](#) (2, 7, 10, 12, 16)
- [2] Martín Abadi and Véronique Cortier. Deciding knowledge in security protocols under equational theories. *Theor. Comput. Sci.* 367(1-2):2–32, 2006. [DOI](#) (4, 9, 14, 19, 20, 75)
- [3] Martín Abadi and Cédric Fournet. Private authentication. *Theor. Comput. Sci.* 322(3):427–476, 2004. [DOI](#) (4, 7, 10)
- [4] Martín Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: the spi calculus. *Inf. Comput.* 148(1):1–70, 1999. [DOI](#) (2, 16)
- [5] Ben Adida. Helios: web-based open-audit voting. *Proceedings of the 17th USENIX Security Symposium, July 28-August 1, 2008, San Jose, CA, USA*, pages 335–348. USENIX Association, 2008. [URL](#) (4, 41)
- [6] Myrto Arapinis, Tom Chothia, Eike Ritter, and Mark Ryan. Analysing unlinkability and anonymity using the applied pi calculus. *Proceedings of the 23rd IEEE Computer Security Foundations Symposium, CSF 2010, Edinburgh, United Kingdom, July 17-19, 2010*, pages 107–121. IEEE Computer Society, 2010. [DOI](#) (2)
- [7] Myrto Arapinis, Véronique Cortier, and Steve Kremer. When are three voters enough for privacy properties? *Computer Security - ESORICS 2016 - 21st European Symposium on Research in Computer Security, Heraklion, Greece, September 26-30, 2016, Proceedings, Part II*, volume 9879 of *Lecture Notes in Computer Science*, pages 241–260. Springer, 2016. [DOI](#) (16)
- [8] Myrto Arapinis, Loretta Ilaria Mancini, Eike Ritter, Mark Ryan, Nico Golde, Kevin Redon, and Ravishankar Borgaonkar. New privacy issues in mobile telephony: fix and verification. *the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012*, pages 205–216. ACM, 2012. [DOI](#) (6, 41)

- [9] Alessandro Armando, David A. Basin, Yohan Boichut, Yannick Chevalier, Luca Compagna, Jorge Cuéllar, Paul Hankes Drielsma, Pierre-Cyrille Héam, Olga Kouchnarenko, Jacopo Mantovani, Sebastian Mödersheim, David von Oheimb, Michaël Rusinowitch, Judson Santiago, Mathieu Turuani, Luca Viganò, and Laurent Vigneron. The AVISPA tool for the automated validation of internet security protocols and applications. *Computer Aided Verification, 17th International Conference, CAV 2005, Edinburgh, Scotland, UK, July 6-10, 2005, Proceedings*, volume 3576 of *Lecture Notes in Computer Science*, pages 281–285. Springer, 2005. DOI (1)
- [10] David Baelde, Stéphanie Delaune, and Lucca Hirschi. Partial order reduction for security protocols. *26th International Conference on Concurrency Theory, CONCUR 2015, Madrid, Spain, September 1-4, 2015*, volume 42 of *LIPICs*, pages 497–510. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015. DOI (4, 5, 40)
- [11] Gilles Barthe, Ugo Dal Lago, Giulio Malavolta, and Itsaka Rakotonirina. Tidy: symbolic verification of timed cryptographic protocols. *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, pages 263–276. ACM, 2022. DOI (85)
- [12] David A. Basin, Jannik Dreier, Lucca Hirschi, Sasa Radomirovic, Ralf Sasse, and Vincent Stettler. A formal analysis of 5g authentication. *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 1383–1396. ACM, 2018. DOI (2, 5, 6)
- [13] David A. Basin, Jannik Dreier, and Ralf Sasse. Automated symbolic proofs of observational equivalence. *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, pages 1144–1155. ACM, 2015. DOI (6)
- [14] Mathieu Baudet. Deciding security of protocols against off-line guessing attacks. *Proceedings of the 12th ACM Conference on Computer and Communications Security, CCS 2005, Alexandria, VA, USA, November 7-11, 2005*, pages 16–25. ACM, 2005. DOI (5)
- [15] Mathieu Baudet. Sécurité des protocoles cryptographiques : aspects logiques et calculatoires. (Security of cryptographic protocols : logical and computational aspects). PhD thesis, École normale supérieure de Cachan, France, 2007. URL (19–21, 30)
- [16] Mathieu Baudet, Véronique Cortier, and Stéphanie Delaune. YAPA: A generic tool for computing intruder knowledge. *ACM Trans. Comput. Log.* 14(1):4:1–4:32, 2013. DOI (14)
- [17] Karthikeyan Bhargavan, Bruno Blanchet, and Nadim Kobeissi. Verified models and reference implementations for the TLS 1.3 standard candidate. *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 483–502. IEEE Computer Society, 2017. DOI (2)
- [18] Bruno Blanchet. Automatic proof of strong secrecy for security protocols. *2004 IEEE Symposium on Security and Privacy (S&P 2004), 9-12 May 2004, Berkeley, CA, USA*, page 86. IEEE Computer Society, 2004. DOI (17)
- [19] Bruno Blanchet. Modeling and verifying security protocols with the applied pi calculus and proverif. *Found. Trends Priv. Secur.* 1(1-2):1–135, 2016. DOI (1)
- [20] Bruno Blanchet, Martín Abadi, and Cédric Fournet. Automated verification of selected equivalences for security protocols. *J. Log. Algebraic Methods Program.* 75(1):3–51, 2008. DOI (6)
- [21] Bruno Blanchet and Ben Smyth. Automated reasoning for equivalences in the applied pi calculus with barriers. *IEEE 29th Computer Security Foundations Symposium, CSF 2016, Lisbon, Portugal, June 27 - July 1, 2016*, pages 310–324. IEEE Computer Society, 2016. DOI (6)
- [22] Bruno Blanchet, Ben Smyth, Vincent Cheval, and Marc Sylvestre. Automatic Cryptographic Protocol Verifier, User Manual and Tutorial. 2020. URL (3, 39)
- [23] Michele Boreale and Luca Trevisan. A complexity analysis of bisimilarity for value-passing processes. *Theor. Comput. Sci.* 238(1-2):313–345, 2000. DOI (3)
- [24] Rohit Chadha, Vincent Cheval, Ștefan Ciobâcă, and Steve Kremer. Automated verification of equivalence properties of cryptographic protocols. *ACM Trans. Comput. Log.* 17(4):23, 2016. DOI (3, 5, 40, 41)
- [25] Vincent Cheval. APTE: an algorithm for proving trace equivalence. *Tools and Algorithms for the Construction and Analysis of Systems - 20th International Conference, TACAS 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014. Proceedings*, volume 8413 of *Lecture Notes in Computer Science*, pages 587–592. Springer, 2014. DOI (5, 41)
- [26] Vincent Cheval and Bruno Blanchet. Proving more observational equivalences with proverif. *Principles of Security and Trust - Second International Conference, POST 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings*, volume 7796 of *Lecture Notes in Computer Science*, pages 226–246. Springer, 2013. DOI (6)

- [27] Vincent Cheval, Hubert Comon-Lundh, and Stéphanie Delaune. Automating security analysis: symbolic equivalence of constraint systems. *Automated Reasoning, 5th International Joint Conference, IJCAR 2010, Edinburgh, UK, July 16-19, 2010. Proceedings*, volume 6173 of *Lecture Notes in Computer Science*, pages 412–426. Springer, 2010. DOI (5)
- [28] Vincent Cheval, Hubert Comon-Lundh, and Stéphanie Delaune. Trace equivalence decision: negative tests and non-determinism. *Proceedings of the 18th ACM Conference on Computer and Communications Security, CCS 2011, Chicago, Illinois, USA, October 17-21, 2011*, pages 321–330. ACM, 2011. DOI (16)
- [29] Vincent Cheval, Véronique Cortier, and Stéphanie Delaune. Deciding equivalence-based properties using constraint solving. *Theor. Comput. Sci.* 492:1–39, 2013. DOI (5, 9, 16, 19–21, 30)
- [30] Vincent Cheval, Raphaëlle Crubillé, and Steve Kremer. Symbolic protocol verification with dice: process equivalences in the presence of probabilities. *35th IEEE Computer Security Foundations Symposium, CSF 2022, Haifa, Israel, August 7-10, 2022*, pages 319–334. IEEE, 2022. DOI (16, 85)
- [31] Vincent Cheval, Steve Kremer, and Itsaka Rakotonirina. DEEPSEC: deciding equivalence properties in security protocols theory and practice. *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*, pages 529–546. IEEE Computer Society, 2018. DOI (1–4, 16)
- [32] Vincent Cheval, Steve Kremer, and Itsaka Rakotonirina. Exploiting symmetries when proving equivalence properties for security protocols. *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, pages 905–922. ACM, 2019. DOI (4, 40)
- [33] Vincent Cheval, Steve Kremer, and Itsaka Rakotonirina. The DEEPSEC prover. *Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part II*, volume 10982 of *Lecture Notes in Computer Science*, pages 28–36. Springer, 2018. DOI (3)
- [34] Vincent Cheval, Steve Kremer, and Itsaka Rakotonirina. The hitchhiker’s guide to decidability and complexity of equivalence properties in security protocols. *Logic, Language, and Security - Essays Dedicated to Andre Scedrov on the Occasion of His 65th Birthday*, volume 12300 of *Lecture Notes in Computer Science*, pages 127–145. Springer, 2020. DOI (20)
- [35] Vincent Cheval, Steve Kremer, Itsaka Rakotonirina, and Victor Yon. Deepsec official website. Installation, manual, tutorial, publications. URL (4, 39)
- [36] Yannick Chevalier and Michaël Rusinowitch. Decidability of equivalence of symbolic derivations. *J. Autom. Reason.* 48(2):263–292, 2012. DOI (5)
- [37] Rémy Chrétien, Véronique Cortier, and Stéphanie Delaune. Decidability of trace equivalence for protocols with nonces. *IEEE 28th Computer Security Foundations Symposium, CSF 2015, Verona, Italy, 13-17 July, 2015*, pages 170–184. IEEE Computer Society, 2015. DOI (5, 8)
- [38] Rémy Chrétien, Véronique Cortier, and Stéphanie Delaune. From security protocols to pushdown automata. *ACM Trans. Comput. Log.* 17(1):3, 2015. DOI (5)
- [39] Ștefan Ciobâcă, Stéphanie Delaune, and Steve Kremer. Computing knowledge in security protocols under convergent equational theories. *J. Autom. Reason.* 48(2):219–262, 2012. DOI (4, 14, 20)
- [40] Katriel Cohn-Gordon, Cas Cremers, Luke Garratt, Jon Millican, and Kevin Milner. On ends-to-ends encryption: asynchronous group messaging with strong security guarantees. *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 1802–1819. ACM, 2018. DOI (2)
- [41] Hubert Comon-Lundh and Stéphanie Delaune. The finite variant property: how to get rid of some algebraic properties. *Term Rewriting and Applications, 16th International Conference, RTA 2005, Nara, Japan, April 19-21, 2005, Proceedings*, volume 3467 of *Lecture Notes in Computer Science*, pages 294–307. Springer, 2005. DOI (5, 23)
- [42] Bruno Concinha, David A. Basin, and Carlos Caleiro. FAST: an efficient decision procedure for deduction and static equivalence. *Proceedings of the 22nd International Conference on Rewriting Techniques and Applications, RTA 2011, May 30 - June 1, 2011, Novi Sad, Serbia*, volume 10 of *LIPICs*, pages 11–20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2011. DOI (4, 14, 20)
- [43] Véronique Cortier, Antoine Dallon, and Stéphanie Delaune. Sat-equiv: an efficient tool for equivalence properties. *30th IEEE Computer Security Foundations Symposium, CSF 2017, Santa Barbara, CA, USA, August 21-25, 2017*, pages 481–494. IEEE Computer Society, 2017. DOI (5, 41)
- [44] Véronique Cortier, Niklas Grimm, Joseph Lallemand, and Matteo Maffei. A type system for privacy properties. *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 409–423. ACM, 2017. DOI (6)
- [45] Cas Cremers, Marko Horvat, Jonathan Hoyland, Sam Scott, and Thyla van der Merwe. A comprehensive symbolic analysis of TLS 1.3. *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 1773–1788. ACM, 2017. DOI (2)

- [46] **Cas J. F. Cremers**. The scyther tool: verification, falsification, and analysis of security protocols. *Computer Aided Verification, 20th International Conference, CAV 2008, Princeton, NJ, USA, July 7-14, 2008, Proceedings*, volume 5123 of *Lecture Notes in Computer Science*, pages 414–418. Springer, 2008. DOI (1)
- [47] **Stéphanie Delaune, Steve Kremer, and Mark Ryan**. Verifying privacy-type properties of electronic voting protocols. *J. Comput. Secur.* 17(4):435–487, 2009. DOI (2)
- [48] **Danny Dolev and Andrew Chi-Chih Yao**. On the security of public key protocols (extended abstract). *22nd Annual Symposium on Foundations of Computer Science, Nashville, Tennessee, USA, 28-30 October 1981*, pages 350–357. IEEE Computer Society, 1981. DOI (2, 7)
- [49] **Nancy A. Durgin, Patrick Lincoln, and John C. Mitchell**. Multiset rewriting and the complexity of bounded security protocols. *J. Comput. Secur.* 12(2):247–311, 2004. DOI (4)
- [50] **Lucca Hirschi, David Baelde, and Stéphanie Delaune**. A method for verifying privacy-type properties: the unbounded case. *IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016*, pages 564–581. IEEE Computer Society, 2016. DOI (6)
- [51] **PKI Task Force**. PKI for machine readable travel documents offering ICC read-only access. Technical report, International Civil Aviation Organization, 2004. (4, 41)
- [52] **Charlie Jacomme and Steve Kremer**. An extensive formal analysis of multi-factor authentication protocols. *31st IEEE Computer Security Foundations Symposium, CSF 2018, Oxford, United Kingdom, July 9-12, 2018*, pages 1–15. IEEE Computer Society, 2018. DOI (2)
- [53] **Nadim Kobeissi, Karthikeyan Bhargavan, and Bruno Blanchet**. Automated verification for secure messaging protocols and their implementations: A symbolic and computational approach. *2017 IEEE European Symposium on Security and Privacy, EuroS&P 2017, Paris, France, April 26-28, 2017*, pages 435–450. IEEE, 2017. DOI (2)
- [54] **Simon Meier, Benedikt Schmidt, Cas Cremers, and David A. Basin**. The TAMARIN prover for the symbolic analysis of security protocols. *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, volume 8044 of *Lecture Notes in Computer Science*, pages 696–701. Springer, 2013. DOI (1, 3)
- [55] **Robin Milner, Joachim Parrow, and David Walker**. A calculus of mobile processes, I. *Inf. Comput.* 100(1):1–40, 1992. DOI (14)
- [56] **Michaël Rusinowitch and Mathieu Turuani**. Protocol insecurity with a finite number of sessions, composed keys is np-complete. *Theor. Comput. Sci.* 299(1-3):451–475, 2003. DOI (4, 5)
- [57] **Peter Y. A. Ryan and Steve A. Schneider**. Prêt à voter with re-encryption mixes. *Computer Security - ESORICS 2006, 11th European Symposium on Research in Computer Security, Hamburg, Germany, September 18-20, 2006, Proceedings*, volume 4189 of *Lecture Notes in Computer Science*, pages 313–326. Springer, 2006. DOI (4, 41)
- [58] **Sonia Santiago, Santiago Escobar, Catherine Meadows, and José Meseguer**. A formal definition of protocol indistinguishability and its verification using Maude-NPA. *Security and Trust Management - 10th International Workshop, STM 2014, Wroclaw, Poland, September 10-11, 2014. Proceedings*, volume 8743 of *Lecture Notes in Computer Science*, pages 162–177. Springer, 2014. DOI (1, 3, 6)
- [59] **Alwen Tiu and Jeremy E. Dawson**. Automating open bisimulation checking for the spi calculus. *Proceedings of the 23rd IEEE Computer Security Foundations Symposium, CSF 2010, Edinburgh, United Kingdom, July 17-19, 2010*, pages 307–321. IEEE Computer Society, 2010. DOI (5)
- [60] **Alwen Tiu, Nam Nguyen, and Ross Horne**. SPEC: an equivalence checker for security protocols. *Programming Languages and Systems - 14th Asian Symposium, APLAS 2016, Hanoi, Vietnam, November 21-23, 2016, Proceedings*, volume 10017 of *Lecture Notes in Computer Science*, pages 87–95, 2016. DOI (5, 41)

A. Decision procedures using partition trees

We detail in this appendix the technical proofs of correctness of the decision procedures for equivalences of Section 3.6, assuming a partition tree T priorly constructed.

A.1 Trace equivalence

We prove in this section the following theorem:

THEOREM 3.20 (partition-tree-based characterisation of trace equivalence). *Whenever $T \in \text{PTree}(P_1, P_2)$, the following points are equivalent:*

1. $P_1 \sqsubseteq_t P_2$
2. for all partition-tree traces $P_1 \xRightarrow{\text{tr}}_T (\mathcal{P}_1, C_1), n$, we have $P_2 \xRightarrow{\text{tr}}_T (\mathcal{P}_2, C_2), n$

The proof of this theorem relies on two technical lemmas extending the properties of the partition tree edges to its branches, i.e., from \rightarrow_T to \Rightarrow_T . For example we can generalise as follows the fact that the nodes of the tree are labelled by maximal configurations, i.e., Definition 3.17, Item 4:

LEMMA A.1. *Assume that $(\mathcal{P}_1, C_1), n \xRightarrow{\text{tr}}_T (\mathcal{P}'_1, C'_1), n'$ and $(\mathcal{P}_2, C_2) \xRightarrow{\text{tr}}_s (\mathcal{P}'_2, C'_2)$ with $(\mathcal{P}_2, C_2) \in \Gamma(n)$. We also consider, for all $i \in \{1, 2\}$, a solution $(\Sigma', \sigma'_i) \in \text{Sol}^{\pi(n')}(C'_i)$ such that $\Phi(C'_1)\sigma'_1 \sim \Phi(C'_2)\sigma'_2$. Then we have $(\mathcal{P}_2, C_2), n \xRightarrow{\text{tr}}_T (\mathcal{P}'_2, C'_2), n'$.*

PROOF. We proceed by induction on the length tr . The case $\text{tr} = \varepsilon$ follows from the saturation of nodes under τ -transition (Definition 3.17, Item 1). Otherwise we let, with $\text{tr} = \alpha \cdot \tilde{\text{tr}}$,

$$(\mathcal{P}_1, C_1), n \xRightarrow{\alpha}_T (\tilde{\mathcal{P}}_1, \tilde{C}_1), \tilde{n} \xRightarrow{\tilde{\text{tr}}}_T (\mathcal{P}'_1, C'_1), n' \quad (\mathcal{P}_2, C_2) \xRightarrow{\alpha}_s (\tilde{\mathcal{P}}_2, \tilde{C}_2) \xRightarrow{\tilde{\text{tr}}}_s (\mathcal{P}'_2, C'_2)$$

We also consider the restrictions $\Sigma = \Sigma'_{|\text{vars}^2(n)}$ and $\tilde{\Sigma} = \Sigma'_{|\text{vars}^2(\tilde{n})}$. In particular $\Sigma \subseteq \tilde{\Sigma}$ and there exist $\sigma_2, \tilde{\sigma}_1, \tilde{\sigma}_2$ such that

$$(\Sigma, \sigma_2) \in \text{Sol}(C_2) \quad (\tilde{\Sigma}, \tilde{\sigma}_1) \in \text{Sol}(\tilde{C}_1) \quad (\tilde{\Sigma}, \tilde{\sigma}_2) \in \text{Sol}(\tilde{C}_2)$$

The hypothesis that $\Phi(C'_1)\sigma'_1 \sim \Phi(C'_2)\sigma'_2$ also implies that $\Phi(\tilde{C}_1)\tilde{\sigma}_1 \sim \Phi(\tilde{C}_2)\tilde{\sigma}_2$. Besides since predicates are refined along branches (Definition 3.17, Item 3) and are defined on the variables of their configurations (Definition 3.16, Item 1), we know that Σ and $\tilde{\Sigma}$ verify $\pi(n)$ and $\pi(\tilde{n})$, respectively.

All in all we can use the maximality of the node \tilde{n} (Definition 3.17, Item 4 applied to the edge $n \xrightarrow{\alpha} \tilde{n}$), which gives that $(\tilde{\mathcal{P}}_2, \tilde{C}_2) \in \Gamma(\tilde{n})$. Hence $(\mathcal{P}_2, C_2), n \xRightarrow{\alpha}_T (\tilde{\mathcal{P}}_2, \tilde{C}_2), \tilde{n}$ by definition and the conclusion then follows from the induction hypothesis applied to the remaining of the traces. ■

Combined with the soundness and the completeness of the symbolic semantics, this permits to prove one direction of Theorem 3.20:

PROOF OF THEOREM 3.20, $1 \Rightarrow 2$. Let us consider a trace $P_1 \xrightarrow{\text{tr}}_T (\mathcal{P}_1, C_1), n$ and exhibit a trace $P_2 \xrightarrow{\text{tr}}_T (\mathcal{P}_2, C_2), n$. We decompose the proof into the following steps:

1. By *soundness* of the symbolic semantics we obtain a trace $P_1 \xrightarrow{\text{tr}\Sigma} (\mathcal{P}_1\sigma_1, \Phi(C_1)\sigma_1\downarrow)$ for an arbitrary solution $(\Sigma, \sigma_1) \in \text{Sol}(C_1)$.
2. By *hypothesis 1* there exists a concrete trace $P_2 \xrightarrow{\text{tr}\Sigma} (\mathcal{P}, \Phi)$ such that $\Phi \sim \Phi(C_1)\sigma_1\downarrow$.
3. By *completeness* of the symbolic semantics we obtain a symbolic trace $P_2 \xrightarrow{\text{tr}'\Sigma}_s (\mathcal{P}_2, C_2)$ and $(\Sigma', \sigma_2) \in \text{Sol}(C_2)$ such that $\text{tr}\Sigma = \text{tr}'\Sigma'$, $\mathcal{P}_2\sigma_2 = \mathcal{P}$ and $\Phi(C_2)\sigma_2\downarrow = \Phi$. Due to the form of symbolic actions, we know that there exists a second-order-variable renaming ϱ such that $\text{tr} = \text{tr}'\varrho$; in particular $P_2 \xrightarrow{\text{tr}}_s (\mathcal{P}_2, C_2\varrho)$ and $(\Sigma, \sigma_2) \in \text{Sol}(C_2\varrho)$.
4. By *Lemma A.1* we therefore obtain that $P_2 \xrightarrow{\text{tr}}_T (\mathcal{P}_2, C_2\varrho), n$, which gives the expected conclusion. ■

The second property of the partition tree we extend is the fact that symbolic transitions are reflected in the tree, i.e., Definition 3.17, Item 2:

LEMMA A.2. *Let n be a node of a partition tree T and $(\mathcal{P}, C) \in \Gamma(n)$. If $(\mathcal{P}, C) \xrightarrow{\text{tr}}_s (\mathcal{P}', C')$ and $(\Sigma, \sigma) \in \text{Sol}^{\pi(n)}(C')$ then there exist a node n' and a substitution Σ' such that $(\mathcal{P}, C), n \xrightarrow{\text{tr}}_T (\mathcal{P}', C'), n'$ and $(\Sigma', \sigma) \in \text{Sol}^{\pi(n')}(C')$.*

Note that unlike the definition of partition tree, we do not require that Σ' coincides with Σ on $\text{vars}^2(n)$. This additional requirement would not make the lemma false but is unnecessary to prove Theorem 3.20. The lemma is proved by induction on tr below:

PROOF. We proceed by induction on the length of tr . If $\text{tr} = \varepsilon$ it suffices to choose $n = n'$ and the conclusion immediately follows. Otherwise let us decompose the symbolic trace into

$$(\mathcal{P}, C) \xrightarrow{\tilde{\text{tr}}}_s (\tilde{\mathcal{P}}, \tilde{C}) \xrightarrow{\alpha}_s (\mathcal{P}', C') \quad \text{tr} = \tilde{\text{tr}} \cdot \alpha$$

Note that $(\Sigma|_{\text{vars}^2(\tilde{n})}, \sigma|_{\text{vars}^1(\tilde{n})}) \in \text{Sol}(\tilde{C})$, and $\Sigma|_{\text{vars}^2(\tilde{n})}$ verifies $\pi(n)$ by definition of a configuration (since Σ verifies it and has the same restriction to $\text{vars}^2(\Gamma(n))$ as $\Sigma|_{\text{vars}^2(\tilde{n})}$). By induction hypothesis we therefore obtain $\tilde{n}, \tilde{\Sigma}$ such that $(\mathcal{P}, C), n \xrightarrow{\tilde{\text{tr}}}_T (\tilde{\mathcal{P}}, \tilde{C}), \tilde{n}$ and $(\tilde{\Sigma}, \sigma|_{\text{vars}^1(\tilde{n})}) \in \text{Sol}^{\pi(\tilde{n})}(\tilde{C})$. Let us then consider the extension

$$\tilde{\Sigma}^e = \tilde{\Sigma} \cup \Sigma|_{\text{vars}^2(n') \setminus \text{vars}^2(\tilde{n})}$$

To conclude the proof it suffices to apply the Item 2 of Definition 3.17 to the symbolic transition $(\tilde{\mathcal{P}}, \tilde{C}) \xrightarrow{\alpha}_s (\mathcal{P}', C')$ and the solution $(\tilde{\Sigma}^e, \sigma)$; what remains to prove is therefore that we effectively have $(\tilde{\Sigma}^e, \sigma) \in \text{Sol}^{\pi(n')}(C')$. First of all we indeed have by construction $\text{dom}(\tilde{\Sigma}^e) = \text{vars}^2(C')$ and $\text{dom}(\sigma) = \text{vars}^1(C')$. We also know that $\tilde{\Sigma}^e$ satisfies the predicate $\pi(n')$ because $\tilde{\Sigma} = \tilde{\Sigma}^e|_{\text{vars}^2(\tilde{n})}$

satisfies it. The first-order solution σ satisfies the constraints of $E^1(C')$ since $(\Sigma, \sigma) \in \text{Sol}(C')$ by hypothesis. Finally we let $\varphi \in D(C')$ and prove that $(\Phi(C'), \tilde{\Sigma}^e, \sigma) \models \varphi$:

▷ *case 1*: $\varphi = (X \vdash^? x) \in D(\tilde{C})$

The conclusion follows from the fact that $(\tilde{\Sigma}, \sigma_{|\text{vars}^1(\tilde{n})}) \in \text{Sol}(\tilde{C})$.

▷ *case 2*: $\varphi = (X \vdash^? x) \in D(C') \setminus D(\tilde{C})$

The conclusion follows from the fact that $(\Sigma, \sigma) \in \text{Sol}(C')$.

▷ *case 3*: $\varphi = \forall X. X \not\vdash^? x$

We have to prove that $x\sigma$ is not deducible from the frame $\Phi(C')\sigma$, which is a consequence from the fact that $(\Sigma, \sigma) \in \text{Sol}(C')$. ■

Using again the soundness and completeness of the symbolic semantics, we can finally derive the other direction of Theorem 3.20.

PROOF OF THEOREM 3.20, $2 \Rightarrow 1$. Let us consider a trace $P_1 \xRightarrow{\text{tr}} (\mathcal{P}, \Phi)$ and exhibit a trace $P_2 \xRightarrow{\text{tr}} (\mathcal{Q}, \Psi)$ such that $\Phi \sim \Psi$. We decompose the proof into the following steps:

1. By *completeness* of the symbolic semantics we obtain a symbolic trace $P_1 \xRightarrow{\text{tr}_s} (\mathcal{P}_1, C_1)$ and $(\Sigma, \sigma_1) \in \text{Sol}(C)$ such that $\text{tr}_s \Sigma = \text{tr}$, $\mathcal{P}_1 \sigma_1 = \mathcal{P}$ and $\Phi(C_1) \sigma_1 \downarrow = \Phi$.
2. By *Lemma A.2* we then obtain a partition-tree trace $P_1 \xRightarrow{\text{tr}_s} (\mathcal{P}_1, C_1), n$ and Σ' such that $(\Sigma', \sigma_1) \in \text{Sol}^{\pi(n)}(C_1)$.
3. By *hypothesis 2* there also exists a partition-tree trace $P_2 \xRightarrow{\text{tr}_s} (\mathcal{P}_2, C_2), n$. By definition of a configuration we also know that there exists σ_2 such that $(\Sigma', \sigma_2) \in \text{Sol}^{\pi(n)}(C_2)$ and $\Phi(C_1) \sigma_1 \sim \Phi(C_2) \sigma_2$.
4. By *soundness* of the symbolic semantics applied to we then obtain a concrete trace $P_2 \xRightarrow{\text{tr}_s \Sigma'} (\mathcal{Q}, \Psi)$ with $\mathcal{Q} = \mathcal{P}_2 \sigma_2$ and $\Psi = \Phi(C_2) \sigma_2 \downarrow \sim \Phi(C_1) \sigma_1 \downarrow = \Phi$.

However we may have $\text{tr}_s \Sigma' \neq \text{tr}$ and, to conclude the proof, we prove that $P_2 \xRightarrow{\text{tr}} (\mathcal{Q}, \Psi)$ as well. For that it suffices to prove that $\text{tr} \Psi \downarrow = \text{tr}_s \Sigma' \Psi \downarrow$, that is, although the recipes or tr and $\text{tr}_s \Sigma'$ are different they produce the same first-order terms. Since Φ and Ψ are statically equivalent, $\text{tr} = \text{tr}_s \Sigma$ and $\Phi = \Phi(C_1) \sigma_1 \downarrow$, it suffices to prove that $\text{tr}_s \Sigma \Phi(C_1) \sigma_1 \downarrow = \text{tr}_s \Sigma' \Phi(C_1) \sigma_1 \downarrow$. Let $X \in \text{vars}^2(\text{tr}_s)$. A quick look at the rules of the symbolic semantics shows that there exists a deduction fact $(X \vdash^? x) \in D(C_1)$. In particular, since (Σ, σ_1) and (Σ', σ_1) are both solutions of C_1 we have $X \Sigma \Phi(C_1) \sigma_1 \downarrow = x \sigma_1 \downarrow = X \Sigma' \Phi(C_1) \sigma_1 \downarrow$, hence the conclusion. ■

A.2 Simulations

In this section, we now prove the main theorem at the basis of the decision procedure for simulations and its variants:

THEOREM 3.25 (partition-tree-based characterisation of labelled bisimilarity). *If $T \in \text{PTree}(P_1, P_2)$:*

1. $P_1 \approx_b P_2$ iff for all symbolic witnesses of non-bisimilarity w_s for $(P_1, P_2, \text{root}(T))$, we have $\text{Sol}(w_s) = \emptyset$
2. $P_1 \sqsubseteq_s P_2$ iff for all symbolic witnesses of non-simulation w_s for $(P_1, P_2, \text{root}(T))$, we have $\text{Sol}(w_s) = \emptyset$

We only prove the case of labelled bisimilarity, as the proof for simulation is analogue. For that, we prove the following technical lemma by induction on the structure of the (symbolic) witness; this lemma is a stronger version of the theorem for the purpose of managing the induction invariant.

LEMMA A.3. *Let n be a node of a partition tree T and $A_0, A_1 \in \Gamma(n)$. We let $A_i = (\mathcal{P}_i, C_i)$ and $\Sigma, \sigma_0, \sigma_1$ such that $(\Sigma, \sigma_i) \in \text{Sol}^{\pi(n)}(C_i)$. If $A_i^c = (\mathcal{P}_i \sigma_i, \Phi(C_i) \sigma_i \downarrow)$, the following points are equivalent:*

1. $A_0^c \not\approx_b A_1^c$
2. *there exist a symbolic witness w_s for (A_0, A_1, n) and a solution $f_{\text{sol}} \in \text{Sol}(w_s)$ such that $f_{\text{sol}}(\text{root}(w_s)) = \Sigma$*

To prove this lemma we first observe that, by definition of a configuration (Definition 3.16), $A_0^c \sim A_1^c$ because these two processes are obtained by instantiating two symbolic processes from a same node n with a common solution Σ . We then prove the two directions separately.

PROOF OF LEMMA A.3, $1 \Rightarrow 2$.

We prove the result by induction on $|\mathcal{P}_0, \mathcal{P}_1|$. The conclusion is immediate if $|\mathcal{P}_0, \mathcal{P}_1| = 0$ as it yields a contradiction: the multisets \mathcal{P}_0 and \mathcal{P}_1 can only contain null processes and the fact that $A_0^c \sim A_1^c$ justifies that $A_0^c \approx_b A_1^c$. Otherwise we let by Proposition 3.22 a witness w of (A_0^c, A_1^c) . Thus, by definition, there exist $b \in \{0, 1\}$ and a transition $A_b^c \xrightarrow{\alpha} A_b'^c = (Q, \Phi)$ such that for all traces $A_{1-b}^c \xrightarrow{\bar{\alpha}} A_{1-b}'^c$ such that $A_0'^c \sim A_1'^c$, we have $(A_0'^c, A_1'^c) \in w$ (and therefore $A_0'^c \not\approx_b A_1'^c$ by Proposition 3.22). Let us now construct a symbolic witness w_s of (A_0, A_1, n) and a suitable solution f_{sol} .

▷ *case 1: $\alpha \neq \tau$*

By completeness of the symbolic semantics (Proposition 3.15) applied to the transition $A_b^c \xrightarrow{\alpha} A_b'^c$, we let a symbolic transition $A_b \xrightarrow{\alpha_s} A_b' = (Q_s, C)$ and a solution $(\Sigma', \sigma') \in \text{Sol}(C)$ such that $\Sigma \subseteq \Sigma'$, $\alpha = \alpha_s \Sigma'$, $Q = Q_s \sigma'$ and $\Phi = \Phi(C) \sigma' \downarrow$. Note that by hypothesis Σ verifies $\pi(n)$ and, therefore, so does its extension Σ' (since by definition predicates are stable by domain extension, recall Definition 3.16). Then since the symbolic transition $A_b \xrightarrow{\alpha_s} A_b'$ is reflected in T (in the sense of Definition 3.17, Item 2), we obtain a transition $A_b, n \xrightarrow{\alpha_s} A_b', n'$ and Σ'' such that $(\Sigma'', \sigma') \in \text{Sol}^{\pi(n')}(C)$ and $\Sigma''|_{\text{vars}^2(n)} = \Sigma'|_{\text{vars}^2(n)} (= \Sigma)$.

▷ *case 1a*: there exist no A'_{1-b} such that $A_{1-b}, n \xrightarrow{\alpha_s} A'_{1-b}, n'$

Then we define w_s to be the tree whose root is labelled (A_0, A_1, n) and that has a unique child labelled (A'_b, n') . We then consider f_{sol} mapping the child to Σ'' and the root to $\Sigma''_{|\text{vars}^2(n)} = \Sigma$, which is a solution of w_s .

▷ *case 1b*: otherwise

In this case we define w_s as follows. Its root is labelled (A_0, A_1, n) and its children are all the nodes labelled (A'_0, A'_1, n') , with $A_{1-b}, n \xrightarrow{\alpha_s} A'_{1-b}, n'$. For each such node, as explained in the beginning of the proof we have $A'_0 \not\approx_b A'_1$ which permits to apply the induction hypothesis with the solution Σ'' . This gives a symbolic witness rooted in this node and f_{sol} a solution mapping this node to Σ'' . Let us write more explicitly these witnesses w_s^1, \dots, w_s^p and $f_{\text{sol}}^1, \dots, f_{\text{sol}}^p$ the corresponding solutions. To conclude it then suffices to choose w_s^1, \dots, w_s^p as the children of the root of w_s , and f_{sol} maps the root of w_s to $\Sigma''_{|\text{vars}^2(n)} = \Sigma$ and each node n of w_s^i to $f_{\text{sol}}^i(n)$.

▷ *case 2*: $\alpha = \tau$

Analogue to case 1 in the simpler case where $n = n'$ and $\Sigma = \Sigma' = \Sigma''$. Note also that the analogue of case 1a cannot arise. ■

PROOF OF LEMMA A.3, $2 \Rightarrow 1$.

We construct a concrete witness w of (A_0^c, A_1^c) as follows:

$$w = \left\{ ((\mathcal{P}_0\sigma_0, \Phi(C_0)\sigma_0\downarrow), (\mathcal{P}_1\sigma_1, \Phi(C_1)\sigma_1\downarrow)) \left| \begin{array}{l} N \text{ node of } w_s \text{ labelled } ((\mathcal{P}_0, C_0), (\mathcal{P}_1, C_1), n), \\ \forall i \in \{0, 1\}, (f_{\text{sol}}(N), \sigma_i) \in \text{Sol}(C_i) \end{array} \right. \right\}$$

The fact that all $(B_0, B_1) \in w$ verify $B_0 \sim B_1$ follows from Definition 3.16. Then let us consider $((\mathcal{P}_0\sigma_0, \Phi(C_0)\sigma_0\downarrow), (\mathcal{P}_1\sigma_1, \Phi(C_1)\sigma_1\downarrow)) \in w$ using the notations of the construction of w above. By definition of a symbolic witness there exists $b \in \{0, 1\}$ and a transition $(\mathcal{P}_b, C_b), n \xrightarrow{\alpha} (\mathcal{P}'_b, C'_b), n'$ such that:

▷ *case 1*: $N = \text{root}(w_s)$ has a unique child N' labelled $\{(\mathcal{P}'_b, C'_b)\}, n'$

Then consider the concrete transition $(\mathcal{P}_b\sigma_b, \Phi(C_b)\sigma_b\downarrow) \xrightarrow{\alpha f_{\text{sol}}(N')} (\mathcal{P}'_b\sigma'_b, \Phi(C'_b)\sigma'_b\downarrow)$ obtained by soundness of the symbolic semantics (Proposition 3.15) where $(f_{\text{sol}}(N'), \sigma'_b) \in \text{Sol}^{\pi(n')}(C'_b)$. By completeness of the symbolic semantics (which is possible to apply since $f_{\text{sol}}(N) \subseteq f_{\text{sol}}(N')$ by definition of a solution of a symbolic witness) and maximality of the node n' (Definition 3.17, Item 4), there cannot exist any concrete trace of the form

$$(\mathcal{P}_{1-b}\sigma_{1-b}, \Phi(C_{1-b})\sigma_{1-b}\downarrow) \xrightarrow{\alpha f_{\text{sol}}(N')} (\mathcal{P}, \Phi)$$

such that $\Phi \sim \Phi(C'_b)\sigma'_b$, hence the conclusion.

▷ *case 2*: the children of $N = \text{root}(w_s)$ are all the nodes N' labelled $((\mathcal{P}'_0, C'_0), (\mathcal{P}'_1, C'_1), n')$, where $(\mathcal{P}_{1-b}, C_{1-b}), n \xrightarrow{\alpha} (\mathcal{P}'_{1-b}, C'_{1-b}), n'$ (and there is at least one such child)

Let N' be an arbitrary child of N , labelled $((\mathcal{P}'_0, C'_0), (\mathcal{P}'_1, C'_1), n')$ with the above notations. As in the previous case we consider the concrete transition obtained by soundness of the symbolic semantics, $(\mathcal{P}_b\sigma_b, \Phi(C_b)\sigma_b\downarrow) \xrightarrow{\alpha f_{\text{sol}}(N')} (\mathcal{P}'_b\sigma'_b, \Phi(C'_b)\sigma'_b\downarrow)$. Then let us consider a trace of the form

$$(\mathcal{P}_{1-b}\sigma_{1-b}, \Phi(C_{1-b})\sigma_{1-b}\downarrow) \xrightarrow{\alpha f_{\text{sol}}(N')} (\mathcal{P}, \Phi) = A \quad \Phi \sim \Phi(C'_b)\sigma'_b$$

Our goal is to prove that $(A, (\mathcal{P}_b\sigma_b, \Phi(C_b)\sigma_b\downarrow)) \in w$. Using the completeness of the symbolic semantics and the maximality of n' as in the previous case, we obtain a partition-tree trace $(\mathcal{P}_{1-b}, C_{1-b}), n \xrightarrow{\alpha} (\mathcal{P}''_{1-b}, C''_{1-b}), n'$ and $(f_{\text{sol}}(N'), \sigma''_{1-b}) \in \text{Sol}^{\pi(n')}(C''_{1-b})$ with $\mathcal{P} = \mathcal{P}''_{1-b}\sigma''_{1-b}$ and $\Phi = \Phi(C''_{1-b})\sigma''_{1-b}\downarrow$. By hypothesis there therefore exists a node N'' , a child of N , labelled $((\mathcal{P}'_b, C'_b), (\mathcal{P}''_{1-b}, C''_{1-b}), n')$. The conclusion then follows from the fact that $f_{\text{sol}}(N') = f_{\text{sol}}(N'')$ by definition of a solution of a symbolic witness. ■

B. Correctness of the generation of partition trees

B.1 Invariants of the procedure

In this section we present some additional properties that are verified all along the procedure by the nodes of the partition tree under construction. Such nodes are sets of extended symbolic processes (i.e. tuples (\mathcal{P}, C, C^e) with \mathcal{P} a process, C a constraint system and C^e an extended constraint system). Understanding the technical details of these invariants is not necessary to understand the algorithm itself, however most of our subprocedure (e.g. the generation of most general solutions) are only correct in their context.

Invariant 1: Well-formedness The first invariant is about the shape of the extended constraint systems. Two important properties are that all equations of E^1 and E^2 are trivially satisfiable (they are essentially of the form $x =^? u$ where x appears nowhere else in the constraint system) and that those of E^2 only use terms that can be constructed from the knowledge base (i.e. they are consequences of $K \cup D$).

DEFINITION B.1. We define the predicate Inv_{wf} on extended constraint systems as follows; we have that $\text{Inv}_{wf}((\Phi, D, E^1, E^2, K, F))$ holds when

- Variables in K and F : $\text{vars}^2(K, F) \subseteq \text{vars}^2(D)$
- Equation: $\text{mgu}(E^i) \neq \perp$, $\text{dom}(\text{mgu}(E^i)) \cap \text{vars}^i(D) = \emptyset$, $\text{vars}^i(\text{img}(\text{mgu}(E^i))) \subseteq \text{vars}^i(D)$.
- Solution is consequence: $\text{img}(\text{mgu}(E^2)) \subseteq \text{Conseq}(K \cup D)$.
- Shape of K : For all $\psi = (\xi \vdash^? u) \in K$, $\psi \in F$, $u \notin \mathcal{X}^1$, $u \in \text{st}(\Phi)$ and $\text{st}(\xi) \subseteq \text{Conseq}(K \cup D)$.
- Shape of F : For all $\forall S. H \Leftarrow \varphi \in F(C)$, S is empty and φ only contains syntactic equations as hypothesis, i.e. no deduction facts. Moreover $\text{sst}^2(H) \subseteq \text{Conseq}(K \cup D)$, and if $H = \xi \vdash^? u$ then either $u \in \text{st}(\Phi)$ or there exists a recipe ζ such that $(\zeta, u) \in \text{Conseq}(K \cup D)$.

This invariant is lifted to sets of extended constraint systems in the natural way, i.e. $\text{Inv}_{wf}(S)$ holds *iff* for all $C \in S$, $\text{Inv}_{wf}(C)$ holds.

Invariant 2: Formula soundness The second invariant states that any substitution that satisfies the deduction facts of D and the equalities of E^1 also satisfies all formulas of $K \cup F$. This means that the procedure only adds correct formulas to the constraint system, sometimes under some hypothesis for the formulas of F .

DEFINITION B.2. We define the predicate Inv_{sound} on extended constraint systems as follows; we have that $\text{Inv}_{sound}((\Phi, D, E^1, E^2, K, F))$ holds when for all $\psi \in K \cup F$, for all substitutions Σ, σ , if

$$(\Phi, \Sigma, \sigma) \models D' \wedge E_{=}^1 \quad \text{with } D' = \{\psi' \in D \mid \text{vars}^2(\psi') \subseteq \text{vars}^2(\psi)\}$$

where $E_{=}^1$ is the set of equations of E^1 , then $(\Phi, \Sigma, \sigma) \models \psi$.

Invariant 3: Formula completeness Given a formula $\psi = H \Leftarrow \varphi$ in F , the soundness invariant above states that when some substitutions satisfy φ then they also satisfy the head H . The next invariant can be seen as a kind of converse statement: when some substitutions satisfy the head H , there exists a formula $\psi' \in F$ (not necessarily the same) that has the same head and whose hypotheses are satisfied. This means that the procedure always covers all cases when generating the potential hypotheses of a given head H .

DEFINITION B.3. In this definition, we say that (Φ, Σ, σ) *weakly satisfies* a head H when:

- if $H = \xi \vdash^? u$ then $\text{msg}(\xi \Sigma \Phi \sigma)$, i.e. the recipe ξ leads to a valid message
- if $H = \xi \stackrel{?}{=} \zeta$ then (Φ, Σ, σ) satisfies H in the usual sense, i.e. the recipes ξ, ζ lead to the same valid message.

Given a set of extended processes Γ , the predicate $\text{Inv}_{comp}(\Gamma)$ holds when for all $C_1^e, C_2^e \in \Gamma$, for all $(H \Leftarrow \varphi) \in F(C_1^e)$, for all $(\Sigma, \sigma) \in \text{Sol}(C_2^e)$, if $(\Phi(C_2^e), \Sigma, \sigma)$ weakly satisfies H then there exists $(H' \Leftarrow \varphi') \in F(C_2^e)$ such that $H' \simeq_r H$ and $(\Phi(C_2^e), \Sigma, \sigma) \models \varphi'$.

Invariant 4: Knowledge-base saturation The next invariant states that the knowledge base K is saturated, i.e. that we do not miss solutions by imposing that they are constructed from K (see Definition 4.4).

DEFINITION B.4. We define the predicate Inv_{satur} on extended constraint systems as follows. We have that $\text{Inv}_{satur}(C)$ holds when, considering the minimal k such that $\text{vars}^2(D(C)) \subseteq \mathcal{X}_{\leq k}^2$, for all $f/n \in \mathcal{F}_d$, for all $(\xi_1, u_1), \dots, (\xi_n, u_n) \in \text{Conseq}(K(C))$ such that $\xi_1, \dots, \xi_n \in \mathcal{T}_k^2$, if $f(u_1, \dots, u_n) \downarrow$ is a constructor term then there is $\xi \in \mathcal{T}_k^2$ such that $(\xi, f(u_1, \dots, u_n) \downarrow) \in \text{Conseq}(K(C) \cup D(C))$.

Invariant 5: Preservation of solutions Finally the last invariant states that all the constraint solving performed on the additional data of extended constraint systems (E^2, K, F) preserves the

solutions. That is, in an extended symbolic process (\mathcal{P}, C, C^e) , where C is the constraint system obtained by only collecting constraints during the execution of the process \mathcal{P} (i.e. without additional constraint solving), all solutions of C^e are solutions of C .

DEFINITION B.5. We define the predicate Inv_{sol} defined on extended symbolic processes where $\text{Inv}_{sol}((\mathcal{P}, C, C^e))$ holds when

- for all $i \in \mathbb{N}$, $\text{vars}^2(C) \subseteq \mathcal{X}_{\leq i}^2$ iff $\text{vars}^2(C^e) \subseteq \mathcal{X}_{\leq i}^2$.
- for all $(\Sigma, \sigma) \in \text{Sol}(C^e)$, $(\Sigma|_{\text{vars}^2(C)}, \sigma|_{\text{vars}^1(C)}) \in \text{Sol}(C)$.

We restrict the substitutions to the variables of C since our extended constraint system may introduce new variables (e.g. by applying most general solutions) but all these variables are uniquely defined by the instantiation of the variables of C .

Invariant 6: Component structure Finally we state an invariant on components stating that all of their constraint systems have the same second-order structure. This invariant is preserved during the procedure by the fact that the constraint-solving rules that modify K or E^2 are always applied to the entire components.

DEFINITION B.6. We define the predicate Inv_{str} defined on sets of extended symbolic processes where $\text{Inv}_{sol}(\Gamma)$ holds when for all $(\mathcal{P}_1, C_1, C_1^e), (\mathcal{P}_2, C_2, C_2^e) \in \Gamma$,

- $\text{dom}(\Phi(C_1^e)) = \text{dom}(\Phi(C_2^e))$
- $\text{vars}^2(C_1^e) = \text{vars}^2(C_2^e)$
- $\{\xi \mid (\xi \vdash^? u) \in K(C_1^e)\} = \{\xi \mid (\xi \vdash^? u) \in K(C_2^e)\}$

Overall invariant As we mentioned, all invariants are lifted to sets of extended symbolic processes in the natural way if needed. We refer as

$$\text{Inv}_{all}(\Gamma) = \text{Inv}_{wf}(\Gamma) \wedge \text{Inv}_{sound}(\Gamma) \wedge \text{Inv}_{comp}(\Gamma) \wedge \text{Inv}_{satur}(\Gamma) \wedge \text{Inv}_{sol}(\Gamma) \wedge \text{Inv}_{str}(\Gamma)$$

the invariant of the whole procedure on the nodes of the partition tree (i.e. sets of extended symbolic processes).

B.2 Preservation of the invariants

In this section we prove that the invariants of the procedure stated in Section B.1 are preserved all along the procedure. First of all we prove the case of the case distinction rules:

LEMMA B.7. *Let \mathbb{S} be a set of set of extended symbolic processes such that $\text{Inv}_{all}(\mathbb{S})$. Let $\mathbb{S} \rightarrow \mathbb{S}'$ by applying one case distinction rule and then normalising the result with the simplification rules. We have that $\text{Inv}_{all}(\mathbb{S}')$.*

For the proof we let $\Gamma' \in \mathbb{S}'$ and write $\Gamma \in \mathbb{S}$ the set from which Γ' is originated, i.e. Γ' is one of the sets obtained after applying one case distinction rule to Γ (either the positive or the negative branch) and then normalising with the simplification rules.

PROOF. (Preservation of Inv_{wf} .) Let $C^{e'} = (\Phi, D, E^1, E^2, K, F)$ for some $(\mathcal{P}', C', C^{e'}) \in \Gamma'$. We consider each item of the definition of the predicate (Definition B.1) and show that $C^{e'}$ verifies them.

- ▷ *property 1 (Variables in K and F):* $\text{vars}^2(K, F) \subseteq \text{vars}^2(D)$.

We first observe that this property is preserved by all simplification rules: therefore it suffices to prove that it is preserved by application of case distinction rules. For that we also observe that if C^e is an extended constraint system verifying this property then for all second-order substitutions Σ , $C^e:\Sigma$ verifies it as well (which is precisely why we consider this notation rather than a raw application $C^e\Sigma$). This is sufficient for getting the expected result for Rule (SAT). The case of the negative branches of Rules (REW) and (EQ) are trivial. Regarding their positive branches, we only treat the case of Rule (EQ) since the treatment of (REW) can be obtained by using a similar reasoning on each formulas added by the rule (the sets F_0 in the notations of Rule (REW)). The rule (EQ) does not introduce elements in K and the only second-order variables introduced in F that are not trivially added to D are from recipes ξ_1, ξ_2 such that $\xi_1 \vdash^? u_1, \xi_2 \vdash^? u_2 \in K(C^e)$ for some $(\mathcal{P}, C, C^e) \in \Gamma$. In particular by hypothesis $\text{vars}^2(\xi_1, \xi_2) \subseteq \text{vars}^2(D(C^e)) \subseteq \text{vars}^2(D(C^{e'}))$, hence the conclusion.

- ▷ *property 2 (first and second-order equations):* we have $\text{mgu}(E^i) \neq \perp$, $\text{dom}(\text{mgu}(E^i)) \cap \text{vars}^i(D) = \emptyset$, and $\text{vars}^i(\text{img}(\text{mgu}(E^i))) \subseteq \text{vars}^i(D)$.

The fact that $\text{mgu}(E^i) \neq \perp$ is a direct consequence of the facts that $C^{e'} \neq \perp$ and that $C^{e'}$ is already normalised by the simplification rules of Figure 8, in particular Rules NORM-UNIF and the rules inherited from Figure 7. The remaining properties are simple invariants of the mgu's that are straightforward to verify.

- ▷ *property 3 (Solution is consequence):* $\text{img}(\text{mgu}(E^2)) \subseteq \text{Conseq}(K \cup D)$.

This property comes from the fact that $\text{mgu}(E^2)$ is only modified in the positive branches of the case distinction rules by applying a mgs Σ to the extended constraint systems $C^e \in \Gamma$. A quick induction on the constraint solving relation for computing mgs \rightsquigarrow show that $\text{img}(\Sigma) \subseteq \text{Conseq}(K(C^e) \cup D(C^e))$, hence the conclusion.

- ▷ *property 4 (Shape of K):* for all $\psi = (\xi \vdash^? u) \in K$, $\psi \in F$, $u \notin \mathcal{X}^1$, $u \in \text{st}(\Phi)$ and $\text{st}(\xi) \subseteq \text{Conseq}(K \cup D)$.

The only rule adding a deduction fact to K is the rule (VECT-ADD-CONSEQ); in particular this gives $\psi \in F$. The added deduction facts are of the form $\xi \vdash^? u$ where, for some $(\mathcal{P}, C, C^e) \in \Gamma$, $\xi \vdash^? u \in F(C^e)$ and $(\zeta, u) \notin \text{Conseq}(K(C^e) \cup D(C^e))$ for any recipe ζ . The fact that $\xi \vdash^? u \in F(C^e)$

and $\text{Inv}_{wf}(\Gamma)$ (property 5) justify that $st(\xi) \subseteq \text{Conseq}(K \cup D)$; this justifies that $u \in st(\Phi)$ as well when taking into account that u is not a consequence of $D(C^e)$ (in particular it cannot be a ground constructor term without names otherwise it would even be consequence of the empty set). Finally the property $u \notin \mathcal{X}^1$ also follows from (ζ, u) not being a consequence of $D(C^e)$.

- *property 5 (Shape of F)*: For all $H \Leftarrow \varphi \in F(C^e)$, φ only contains syntactic equations as hypothesis, i.e. no deduction facts. Moreover $sst^2(H) \subseteq \text{Conseq}(K \cup D)$, and if $H = \xi \vdash^? u$ then either $u \in st(\Phi)$ or there exists a recipe ζ such that $(\zeta, u) \in \text{Conseq}(K \cup D)$.

The property that the hypotheses of the formula only contain syntactic equations can be obtained by a straightforward inspection of each case-distinction and simplification rules. Note in particular that whenever a formula ψ with deduction-fact hypotheses is considered (in Rules (REW) and (EQ)), they are applied to a substitution Σ so that $\psi:(\Sigma, C^e)$ only has equations as hypotheses. As for the second part of property 5 we write 1. the property $sst^2(H) \subseteq \text{Conseq}(K \cup D)$ and 2. the rest (the property about the head terms of deduction formulas). We perform a case analysis on the rule that added the formula to Γ' .

- rule (VECT-ADD-FORMULA): the proof of 2 directly follows from $\text{Inv}_{wf}(\Gamma)$ since the rule does not add a deduction formula. Regarding 1, using the notations of the rule, the head of the formula is $\xi =_f^? \zeta$ where $\xi \vdash^? u \in F$ for some u , and $\zeta \in \text{Conseq}(K \cup D)$. In particular the conclusion follows from $\text{Inv}_{wf}(\Gamma)$ (property 4 for the subterms of ξ and property 5 for the subterms of ζ)
- rule (REW): we first prove 1. Using the notations of the rule, all non-root positions of the head of a formula of F_0 are either variables X such that D contains a deduction fact $X \vdash^? x$ (generated by the skeleton), a position of ξ_0 , or a public function symbol (since the rewriting system is constructor, the left-hand sides ℓ of the rewrite rules only contain a destructor at their roots). In particular all strict subterms of H are consequences of $K \cup D$. Now let us prove 2. By definition of the rule, u is a constructor term in normal form obtained after applying one rewrite rule $\ell \rightarrow r$ at the root of $C[u_0]$ for some context C (not containing names but possibly containing variables x such that $X \vdash^? x \in D$ for some X). We recall that the rewriting system is constructor-destructor and subterm convergent, which leaves two cases. The first is that r is a ground constructor term, and then $u = r$ is trivially a consequence of $K \cup D$ for the recipe $\zeta = r$. Otherwise r is a subterm of ℓ , meaning that u is a subterm of $C[u_0]$. This implies that u is either a subterm of u_0 , a subterm of the context C , or a term of the form $C'[u_0]$ for some subcontext C' of C . In all cases, since $u_0 \in st(\Phi)$ by $\text{Inv}_{wf}(\Gamma)$ (property 4), this gives the expected conclusion.
- rule (EQ): the proof of 2 follows from $\text{Inv}_{wf}(\Gamma)$ since the rule does not add a deduction formula. Regarding 1, this follows from $\text{Inv}_{wf}(\Gamma)$ (property 4). ■

PROOF. (Preservation of $\text{Inv}_{\text{sound}}$.) Let $C' = (\Phi, D, E^1, E^2, K, F)$ for some $(\mathcal{P}', C', C^{e'}) \in \Gamma'$, $\psi \in K \cup F$, and (Σ, σ) such that

$$(\Phi, \Sigma, \sigma) \models \bigwedge_{\substack{\psi' \in D \\ \text{vars}^2(\psi') \subseteq \text{vars}^2(\psi)}} \psi' \quad \wedge \quad E^1_{=}$$

We have to prove that $(\Phi, \Sigma, \sigma) \models \psi$. Here we prove more precisely that the conclusion holds when applying one time any of the case-distinction or simplification rules. First of all we observe that this property is preserved by the application of a mgs to an extended constraint system. In particular this makes the conclusion immediate for all rules except the following:

- Rule (REW) (positive branch): using the notations of the rule and recalling $\text{Inv}_{\text{sound}}(\Gamma)$, the conclusion follows from the fact that if $\xi_0 \Sigma \Phi \sigma \downarrow = u_0 \downarrow$, $\ell \rightarrow r \in \mathcal{R}$ and $C[u] = \ell \sigma'$ for some substitution σ' , then $C[\xi_0] \Sigma \Phi \sigma \downarrow = r \sigma' \downarrow$.
- Rule (EQ) (positive branch): using the notations of the rule and recalling $\text{Inv}_{\text{sound}}(\Gamma)$, if ψ is the formula added to F by this rule, we have $(\Phi, \Sigma, \sigma) \models \psi$ iff for some recipes ξ_1, ξ_2 , if $\xi_1 \Sigma \Phi \sigma \downarrow = z$ and $\xi_2 \Sigma \Phi \sigma \downarrow = z$ then $\xi_1 \Sigma \Phi \sigma \downarrow = \xi_2 \Sigma \Phi \sigma \downarrow$. This naturally holds.
- Rule (VECT-ADD-CONSEQ): since the element added to K is originated from F , the conclusion follows from $\text{Inv}_{\text{sound}}(\Gamma)$.
- Rule (VECT-ADD-FORMULA): the reasoning is identical to the one for (EQ). ■

PROOF. (Preservation of Inv_{comp} .) Let $C_1^{e'}, C_2^{e'}$ for some $(\mathcal{P}'_1, C_1, C_1^{e'}), (\mathcal{P}'_2, C_2, C_2^{e'}) \in \Gamma'$, $\psi = H \Leftarrow \varphi \in F(C_1^{e'})$ and $(\Sigma, \sigma) \in \text{Sol}(C_2^{e'})$. We assume that $(\Phi(C_2^{e'}), \Sigma, \sigma)$ weakly satisfies H . We want to prove that there exists $\psi' = (H' \Leftarrow \varphi') \in F(C_2^{e'})$ such that $H' \simeq_r H$ and $(\Phi(C_2^{e'}), \Sigma, \sigma) \models \varphi'$. We prove the strengthened property stating that the invariant is preserved after the application of each case-distinction and simplification rules.

- Rules (NORM-UNIF), (NORM-NO-MGS), (NORM-DISEQ): the conclusion directly follows from $\text{Inv}_{\text{comp}}(\Gamma)$.
- Rule (NORM-FORMULA): by $\text{Inv}_{\text{comp}}(\Gamma)$ we let $\psi'_0 = (H'_0 \Leftarrow \varphi'_0) \in F(C_2^e)$ such that $H'_0 \simeq_r H$ and $(\Phi(C_2^e), \Sigma, \sigma) \models \varphi'_0$. The conclusion directly follows from this, except if ψ'_0 is the formula removed by the rule, i.e. if $\text{mgs}(C_2^e[E^1 \mapsto E^1 \wedge \varphi'_0]) = \emptyset$. However this would yield a contradiction with $(\Phi(C_2^e), \Sigma, \sigma_2) \models \varphi'_0$, hence the conclusion.
- Rule (NORM-DUPL): using the same notations as in the previous case, we let ψ'_0 by $\text{Inv}_{\text{comp}}(\Gamma)$ and the only non-trivial case is again the one where ψ'_0 is removed from C_2^e by the rule. This means that there exists $\psi' \in F(C_2^{e'})$ solved such that $\psi' \simeq_r H'_0$, hence the conclusion since $\text{hyp}(\psi') = \top$ and $H'_0 \simeq_r H$.
- Rules (VECT-RM-UNSAT) and (VECT-SPLIT): the conclusion follows from the $\text{Inv}_{\text{comp}}(\Gamma)$ since $\Gamma' \subseteq \Gamma$.
- Rule (VECT-ADD-CONSEQ): directly follows from $\text{Inv}_{\text{comp}}(\Gamma)$.

— Rule (VECT-ADD-FORMULA): Let us write

$$\Gamma' = \{(\mathcal{P}, C, C^e[F \mapsto F \wedge \psi:(\Sigma, C^e)]) \mid (\mathcal{P}, C, C^e) \in \Gamma\}$$

with the notations and assumptions of the rule. If $i \in \{1, 2\}$ we write $S_i = (\mathcal{P}, C, C_i^e)$. The only case that does not directly follows from $\text{Inv}_{\text{comp}}(\Gamma)$ is the case where $\psi = \psi:(\Sigma, C_1^e)$. But since there exists a formula $\xi \vdash^? u_{S_2} \in F(C_2^e)$ by hypothesis, we know by $\text{Inv}_{\text{sound}}(\Gamma)$ that $\xi\Phi(C_2^e)\Sigma\sigma_2 \downarrow = u_{S_2}$. In particular since $(\Phi(C_2^e), \Sigma, \sigma_2)$ weakly satisfies $\xi =_f^? \zeta$, we can write $u'_{S_2} = \zeta\Sigma\Phi(C_2^e)\sigma_2 \downarrow$ and have $u_{S_2} = u'_{S_2}$. Since we also have by definition (after formula normalisation, see Figure 7)

$$\psi:(\Sigma, C_2^e) = (\xi =_f^? \zeta \Leftarrow u_{S_2} =^? u'_{S_2})$$

we obtain the expected conclusion.

- Any case-distinction rule (negative branch) or Rule (SAT): follows from $\text{Inv}_{\text{comp}}(\Gamma)$.
- Rule (REW) (positive branch): using the notations of the rule, the only case that does not directly follow from the assumption $\text{Inv}_{\text{comp}}(\Gamma)$ is the case where $\psi \in F_0$. The hypothesis $(\Phi(C_2^e), \Sigma, \sigma)$ weakly models the head of ψ can then be rephrased as $\text{msg}(\xi'\Sigma\Phi(C_2^e)\sigma)$ for some recipe $\xi' = C[\xi_0]$ such that $\text{root}(C) \in \mathcal{F}_d$, by definition of F_0 . Let us write $u = \xi_0\Sigma\Phi(C_2^e)\sigma \downarrow$. Since the rewriting system is constructor-destructor and $\text{root}(C[u]) \in \mathcal{F}_d$, there exists at least one rewrite rule $\ell' \rightarrow r' \in \mathcal{R}$ such that $C[u]$ and ℓ' are unifiable. In particular it suffices to choose ψ' the formula of F_0 corresponding to picking this rule in the definition of $\text{RewF}(\xi, \ell \rightarrow r, p)$.
- Rule (EQ) (positive branch): easily follows from $\text{Inv}_{\text{comp}}(\Gamma)$ since the rule only adds to each $(\mathcal{P}, C, C^e) \in \Gamma$ the same solved formula. ■

PROOF. (Preservation of $\text{Inv}_{\text{sat}}.$) Let $C' = (\Phi, D, E^1, E^2, K, F)$ for some $(\mathcal{P}', C', C^{e'}) \in \Gamma'$ and k such that $\text{vars}^2(D(C)) \subseteq \mathcal{X}_{\leq k}^2$. We also let $f/n \in \mathcal{F}_d$ and $(\xi_1, u_1), \dots, (\xi_n, u_n) \in \text{Conseq}(K(C))$ such that $\xi_1, \dots, \xi_n \in \mathcal{T}_k^2$, and $f(u_1, \dots, u_n) \downarrow$ is a constructor term. We have to prove that there is $\xi \in \mathcal{T}_k^2$ such that $(\xi, f(u_1, \dots, u_n) \downarrow) \in \text{Conseq}(K(C) \cup D(C))$. This can be justified by $\text{Inv}_{\text{wf}}(\Gamma)$ (in particular the item “shape of K ”). Indeed, by definition the term $f(u_1, \dots, u_n)$ contains a single destructor, which is the symbol f at its root. In particular if $f(u_1, \dots, u_n) \downarrow$ is indeed a constructor protocol term u , this has been obtained after applying a single rewriting rule at the root of $f(u_1, \dots, u_n)$ since the rewriting system is constructor-destructor. Therefore, by subterm convergence, u is either a ground protocol term (without names, by definition of a rewrite rule) or a subterm of one of the u_i s. In the former case the conclusion is immediate, in the latter it follows from the invariant Inv_{wf} . ■

PROOF. (Preservation of $\text{Inv}_{\text{sol}}.$) The preservation of this invariant is straightforward: the case distinction and simplification rules only modify the extended constraint systems by 1. applying

a mgs, or 2. adding formulas to E^1 , E^2 , K, \dots , or 3. removing formulas from F , or 4. removing trivially-false disequations from E^1 . In particular such operations restrict the set of solutions. ■

We can then extend this property to the whole procedure by proving the preservation by symbolic rules.

LEMMA B.8. *Let Γ be a set of extended symbolic processes such that $\text{Inv}_{all}(\{\Gamma\})$. We assume that no case-distinction or simplification rules can be applied to Γ . We then let*

$$\Gamma_{in} = \{(\mathcal{P}', C', C^{e'}) \mid (\mathcal{P}, C, C^e) \in \Gamma, (\mathcal{P}, C, C^e) \xrightarrow{Y(X)}_s (\mathcal{P}', C', C^{e'})\}$$

$$\Gamma_{out} = \{(\mathcal{P}', C', C^{e'}) \mid (\mathcal{P}, C, C^e) \in \Gamma, (\mathcal{P}, C, C^e) \xrightarrow{\bar{Z}\langle \text{ax}_{|\Phi(C^e)|+1} \rangle}_s (\mathcal{P}', C', C^{e'})\}$$

and the set of set of symbolic processes \mathbb{S} obtained by normalising $\{\Gamma_{in}, \Gamma_{out}\}$ with the simplification rules. Then $\text{Inv}_{all}(\mathbb{S})$.

PROOF. Most invariants are either easily seen to be preserved by application of any symbolic or simplification rules, or are a straightforward consequence of the fact that no simplification rules can be applied to \mathbb{S} . The only substantial case is the Invariant 4 stating that the knowledge base is saturated. Let us then consider $(\mathcal{P}, C, C^e) \in \Gamma'$ for some $\Gamma' \in \mathbb{S}$ and we let k the minimal index such that $\text{vars}^2(D(C)) \subseteq \mathcal{X}_{\leq k}^2$. We then let $f/n \in \mathcal{F}_d$ and $(\xi_1, u_1), \dots, (\xi_n, u_n) \in \text{Conseq}(K(C^e))$ such that $u \downarrow$, with $u = f(u_1, \dots, u_n)$ is a constructor term. We recall that no case-distinction rules can be applied to $\{\Gamma\}$, in particular (REW). But since the rewriting system is constructor-destructor, $f \in \mathcal{F}_d$, and $u \downarrow$ is a constructor term, this means that a rewrite rule is applicable at the root of u . We distinguish two cases.

- *case 1:* there exists a rule $\ell \rightarrow r$ applicable at the root of u , $i \in \llbracket 1, n \rrbracket$ and p a position such that $\ell_{|i \cdot p} \notin \mathcal{X}^1$ and $(\xi_{i|p} \vdash^? v) \in K(C^e)$ for some v .

We consider the instance of Rule (REW) with the following parameters: the position $i \cdot p$, the rule $\ell \rightarrow r$, a recipe $\xi \in \mathcal{T}(\mathcal{F}, \mathcal{X}:|\Phi(C^e)|)$, the formula ψ_0 obtained by considering the rule $\ell \rightarrow r$ in $\text{RewF}(\xi, \ell \rightarrow r, i \cdot p)$, the deduction fact $\xi_{i|p} \vdash^? v$, $\Sigma_0 = \{\xi_{i \cdot p} \mapsto \xi_{i|p}\}$, and a mgs Σ such that the head of $\psi_0: (\Sigma_0 \Sigma, C^e: \Sigma)$ is of the form $\zeta \vdash^? u \downarrow$ for some recipe ζ . Since this instance of the rule is not applicable by hypothesis, we deduce that there already exists a solved formula $\psi \in F(C^e)$ of the form $\zeta' \vdash^? u \downarrow$. Since no simplification rules are applicable neither, in particular (VECT-ADD-CONSEQ), we deduce that there exists a recipe ξ' such that $(\xi', u \downarrow) \in \text{Conseq}(K(C^e) \cup D(C^e))$.

- *case 2:* otherwise

We let $\ell \rightarrow r$ an arbitrary rewrite rule applicable at the root of u . By hypothesis for all positions $i \cdot p$ of ℓ that are not variables we know that $\xi_{i|p}$ is not the recipe of a deduction fact from $K(C^e)$; since $\xi_i \in \text{Conseq}(K(C^e))$, $\text{root}(\xi_{i|p})$ is therefore a constructor symbol. We rule

out the immediate case where r is a ground term and only consider the one where r is a strict subterm of ℓ . Then, since the rewriting system is constructor-destructor, we can fix a ground constructor context C such that $r = C[x_p, \dots, x_q]$ for x_p, \dots, x_q the variables numbered p to q of ℓ (w.r.t. the lexicographic ordering on the positions of the term ℓ). In particular the recipe $(C[\xi_p, \dots, \xi_q], u \downarrow) \in \text{Conseq}(\mathcal{K}(C^e))$. ■

B.3 Preliminary technical results

In this section we prove some low level technical results that will be useful during the incoming proofs.

Preservation by application of substitutions In this section, we show that applying substitution preserves in some cases the different notions we use in our algorithms, namely first-order and second-order equations, deduction and equality facts; and uniformity.

LEMMA B.9. *Let ψ be either a deduction fact, or an equality fact or a first-order equation or a second-order equation. For all ground frame Φ , for all substitutions $\Sigma, \Sigma', \sigma, \sigma'$ if $\text{dom}(\Sigma) \cap \text{dom}(\Sigma') = \emptyset$ and $\text{dom}(\sigma) \cap \text{dom}(\sigma') = \emptyset$ then $(\Phi, \Sigma\Sigma', \sigma\sigma') \models \psi$ is equivalent to $(\Phi, \Sigma\Sigma', \sigma\sigma') \models \psi\Sigma\sigma$ and is equivalent to $(\Phi, \Sigma', \sigma') \models \psi\Sigma\sigma$.*

PROOF. The proof of this lemma is done by case analysis on ψ .

Case $u =^? v$: Consider $(\Phi, \Sigma\Sigma', \sigma\sigma') \models u =^? v$. This is equivalent to $u\sigma\sigma' = v\sigma\sigma'$. Since $\text{vars}(\Sigma) \cap \mathcal{X}^1 = \emptyset$, we deduce that $(\Phi, \Sigma\Sigma', \sigma\sigma') \models u =^? v$ is equivalent to $u\Sigma\sigma\sigma' = v\Sigma\sigma\sigma'$. This is also equivalent to $u\Sigma\sigma\sigma\sigma' = v\Sigma\sigma\sigma\sigma'$ and so $(\Phi, \Sigma\Sigma', \sigma\sigma') \models u\Sigma\sigma =^? v\Sigma\sigma$. Note that $u\Sigma\sigma\sigma' = v\Sigma\sigma\sigma'$ is also equivalent to $(\Phi, \Sigma', \sigma') \models u\Sigma\sigma =^? v\Sigma\sigma$.

Case $\xi =^? \xi'$: Similar to the previous case.

Case $\xi \vdash^? u$: Consider $(\Phi, \Sigma\Sigma', \sigma\sigma') \models \xi \vdash^? u$. It is equivalent to $\xi\Sigma\Sigma'\Phi \downarrow = u\sigma\sigma'$ and $\text{msg}(\xi\Sigma\Sigma'\Phi)$. But $(\xi \vdash^? u)\Sigma\sigma = \xi\Sigma \vdash^? u\sigma$. Moreover, by definition of substitution (in particular the acyclic property), we deduce that $\xi\Sigma\Sigma'\Phi = \xi\Sigma\Sigma\Sigma'\Phi$ and $u\sigma\sigma' = u\sigma\sigma\sigma'$. Hence, $\text{msg}(\xi\Sigma\Sigma'\Phi)$ is equivalent to $\text{msg}(\xi\Sigma\Sigma\Sigma'\Phi)$; and $\xi\Sigma\Sigma'\Phi \downarrow = u\sigma\sigma'$ is equivalent to $\xi\Sigma\Sigma\Sigma'\Phi \downarrow = u\sigma\sigma\sigma'$. Hence $(\Phi, \Sigma\Sigma', \sigma\sigma') \models \xi \vdash^? u$ is equivalent to $(\Phi, \Sigma\Sigma', \sigma\sigma') \models \xi \vdash^? u\Sigma\sigma$. Note that $\text{msg}(\xi\Sigma\Sigma'\Phi)$ and $\xi\Sigma\Sigma'\Phi \downarrow = u\sigma\sigma'$ are also equivalent to $(\Phi, \Sigma', \sigma') \models \xi\Sigma \vdash^? u\sigma$.

Case $\xi =^?_f \xi'$: Similar to previous case. ■

Properties on consequence of set of deduction facts This section contains some results about the consequence relation when modifying a set of deduction facts. The first lemma is about the application of substitutions.

LEMMA B.10. *Let S be a set of solved deduction facts. For all substitutions σ of protocol terms, for all $(\xi, t) \in \text{Conseq}(S)$, $(\xi, t\sigma) \in \text{Conseq}(S\sigma)$.*

PROOF. We know that $(\xi, t) \in \text{Conseq}(S)$ implies $\xi = C[\xi_1, \dots, \xi_n]$ and $t = C[t_1, \dots, t_n]$ for some public context C and for all i , $(\xi_i \vdash^? t_i) \in S$. Hence $(\xi_i \vdash^? t_i)\sigma \in S\sigma$ which allows us to conclude. ■

PROPOSITION B.11 (transitivity of consequences). *Let S, S' be two sets of solved deduction facts. Let $\varphi = \{X_i \vdash^? u_i\}_{i=1}^n$ such that all X_i are pairwise distinct, let $(\xi, t) \in \text{Conseq}(S \cup \varphi)$ and Σ, σ be two substitutions. If for all $i \in \llbracket 1, n \rrbracket$, $(X_i\Sigma, u_i\sigma) \in \text{Conseq}(S\Sigma\sigma \cup S')$ then $(\xi\Sigma, t\sigma) \in \text{Conseq}(S\Sigma\sigma \cup S')$.*

PROOF. We prove this result by induction on $|\xi|$. The base case ($|\xi| = 0$) is trivial as there are no terms of size 0 and we hence focus on the inductive step. We perform a case analysis on the hypothesis $(\xi, t) \in \text{Conseq}(S \cup \varphi)$.

▷ *case 1: $\xi = t \in \mathcal{F}_0$*

We directly have by definition that $(\xi\Sigma, t\sigma) \in \text{Conseq}(S\Sigma\sigma \cup S')$.

▷ *case 2: there are $\xi_1, t_1, \dots, t_m, \xi_m$ and $f \in \mathcal{F}_c$ such that $\xi = f(\xi_1, \dots, \xi_m)$, $t = f(t_1, \dots, t_m)$ and for all $i \in \llbracket 1, m \rrbracket$, $(\xi_i, t_i) \in \text{Conseq}(S \cup \varphi)$*

By induction hypothesis we know that for all $j \in \llbracket 1, m \rrbracket$, $(\xi_j\Sigma, t_j\sigma) \in \text{Conseq}(S\Sigma\sigma \cup S')$. Writing $\xi\Sigma = f(\xi_1\Sigma, \dots, \xi_m\Sigma)$ and $t\sigma = f(t_1\sigma, \dots, t_m\sigma)$, we conclude that $(\xi\Sigma, t\sigma) \in \text{Conseq}(S\Sigma\sigma \cup S')$.

▷ *case 3: $\xi \vdash^? t \in S \cup \varphi$*

If $(\xi \vdash^? t) \in S$ then $(\xi\Sigma \vdash^? t\sigma) \in S\Sigma\sigma$ and the result directly holds. Otherwise $\xi \vdash^? t \in \varphi$ and hence by hypothesis $(\xi\Sigma, t\sigma) \in \text{Conseq}(S\Sigma\sigma \cup S')$. ■

The previous lemma showed that a consequence (ξ, t) is preserved when applying some substitution Σ, σ under the right conditions. However, it is quite strong since we ensure that $\xi\Sigma$ is consequence with $t\sigma$. In some cases, we cannot guarantee that $\xi\Sigma$ is consequence with $t\sigma$ but with some other first-order term. This is the purpose of the next lemma.

LEMMA B.12. *Let S, S' be two sets of solved deduction facts. Let $\varphi = \{X_i \vdash^? u_i\}_{i=1}^n$ such that all X_i are pairwise distinct. For all Σ , for all $\xi \in \text{Conseq}(S \cup \varphi)$, if for all $i \in \{1, \dots, n\}$, $X_i\Sigma \in \text{Conseq}(S\Sigma \cup S')$ then $\xi\Sigma \in \text{Conseq}(S\Sigma \cup S')$.*

PROOF. We prove this result by induction on $|\xi|$. The base case ($|\xi| = 0$) being trivial as there is no term of size 0, we focus on the inductive step.

Since ξ is consequence of $S \cup \varphi$, we know by definition that there exists t such that one of the following conditions hold:

1. $\xi = t \in \mathcal{F}_0$
2. there exists $\xi_1, t_1, \dots, t_m, \xi_m$ and $f \in \mathcal{F}_c$ such that $\xi = f(\xi_1, \dots, \xi_m)$, $t = f(t_1, \dots, t_m)$ and for all $i \in \{1, \dots, m\}$, (ξ_i, t_i) is consequence of $S \cup \varphi$.

3. there exists t such that $\xi \vdash^? t \in S \cup \varphi$.

In case 1, we directly have by definition that $(\xi\Sigma, t)$ is a consequence of $S\Sigma \cup S'$. In case 2, by our inductive hypothesis on ξ_1, \dots, ξ_m , we have that for all $j \in \{1, \dots, m\}$, $\xi_j\Sigma$ is a consequence of $S\Sigma \cup S'$ hence there exists t'_1, \dots, t'_m such that for all $j \in \{1, \dots, m\}$, $(\xi_j\Sigma, t'_j)$ is a consequence of $S\Sigma \cup S'$. With $\xi\Sigma = f(\xi_1\Sigma, \dots, \xi_m\Sigma)$ and $t' = f(t'_1, \dots, t'_m)$, we conclude that $(\xi\Sigma, t')$ is consequence of $S\Sigma \cup S'$. In case 3, if $\xi \vdash^? t \in S$ then $\xi\Sigma \vdash^? t \in S\Sigma$ and so the result directly holds. Else $\xi \vdash^? t \in \varphi$ and so by hypothesis $\xi\Sigma \in \text{Conseq}(S\Sigma \cup S')$. ■

In the next lemma, we show that when a recipe is consequence of the sets of solved deduction formulas $K\Sigma\sigma$ where (Σ, σ) is a solution of the constraint system, then all subterms of that recipe are also consequence of $K\Sigma\sigma$. This property is in fact guaranteed by the fact that K contains itself recipes consequence of itself. This is an important property that allows us to generate solutions that satisfy the uniformity property.

LEMMA B.13. *Let $C = (\Phi, D, E^1, E^2, K, F)$ be an extended constraint system such that $\text{Inv}_{wf}(C)$. For all $\xi \in \text{Conseq}(K \cup D)$, $st(\xi) \subseteq \text{Conseq}(K \cup D)$.*

PROOF. Since $\xi \in \text{Conseq}(K \cup D)$, we know that $\xi = C[\xi_1, \dots, \xi_n]$ where C is a public context and ξ_1, \dots, ξ_n are recipes of deduction facts from K or D . Hence since $\xi' \in st(\xi)$, we have that the position p of ξ' in ξ is either a position of C thus $\xi' \in \text{Conseq}(K \cup D)$ from the definition of consequence; or is a position of one of the ξ_i and thus we conclude by the predicate $\text{Inv}_{wf}(C)$. ■

LEMMA B.14. *Let $C = (\Phi, D, E^1, E^2, K, F)$ be an extended constraint system such that $\text{Inv}_{wf}(C)$. For all $(\Sigma, \sigma) \in \text{Sol}(C)$, for all $\xi \in \text{Conseq}(K\Sigma\sigma)$, $st(\xi) \subseteq \text{Conseq}(K\Sigma\sigma)$.*

LEMMA B.15. *Let S be a set of ground deduction facts. Let Φ be a ground frame. Assume that for all $\psi \in S$, $\Phi \models \psi$. For all $(\xi, t) \in \text{Conseq}(S)$, $\Phi \models \xi \vdash^? t$.*

PROOF. We prove this result by induction on $|\xi|$. The base case being trivial, we focus on the inductive step. Since (ξ, t) is consequence of S then one of the following properties holds:

1. $\xi = t \in \mathcal{F}_0$
2. there exist $\xi_1, t_1, \dots, \xi_n, t_n$ and $f \in \mathcal{F}_c$ such that $\xi = f(\xi_1, \dots, \xi_n)$, $t = f(t_1, \dots, t_n)$ and for all $i \in \{1, \dots, n\}$, (ξ_i, t_i) is consequence of S
3. $\xi \vdash^? t \in S$.

In Case 1, the result trivially holds. In case two, a simple induction on $(\xi_1, t_1), \dots, (\xi_n, t_n)$ allows us to conclude. In case 3, we know by hypothesis that $\Phi \models \xi \vdash^? t$ hence the result holds. ■

B.4 Correctness of most general solutions

In this section we prove the correctness of the constraint solving procedure for computing mgs' in Section 4.2. We show that given an extended constraint system C , the Rules (MGS-CONSEQ),

(MGS-RES), (MGS-CONS), (MGS-UNSAT), and (MGS-UNIF) allow to compute the most general solutions of C .

LEMMA B.16. *Let C an extended constraint system such that $\text{Inv}_{wf}(C)$ and $\text{Inv}_{sound}(C)$. If any rule is applicable on C then for all $(\Sigma, \sigma) \in \text{Sol}(C)$, there exists C', Σ' such that $C \Rightarrow_{\xi} C'$ and $(\Sigma\Sigma', \sigma) \in \text{Sol}(C')$.*

PROOF. First, assume that there exist $\xi, \zeta \in R(C)^2$ and u such that $\xi \neq \zeta$ and $(\xi, u), (\zeta, u) \in \text{Conseq}(K \cup D)$. By $\text{Inv}_{sound}(C)$ and Lemmas B.15 and Proposition B.11, we deduce that $\Phi\sigma \models \xi\Sigma \vdash^? u\sigma \wedge \zeta\Sigma \vdash^? u\sigma$. As such we have $\xi\Sigma \downarrow = u\sigma = \zeta\Sigma \downarrow$. However by definition of a solution it implies that $\xi\Sigma = \zeta\Sigma$. Thus there exists $\Sigma' = \text{mgu}(\xi =^? \zeta)$ such that $\Sigma' \neq \emptyset$ and $\Sigma' \neq \perp$.

In such a case, let us show that $(\Sigma, \sigma) \in \text{Sol}(C')$ with $C \rightarrow C'$ by Rule (MGS-CONSEQ). We already know that $\Sigma \models E^2(C)$ and since $\xi\Sigma = \zeta\Sigma$ with $\Sigma' = \text{mgu}(\xi =^? \zeta)$, we directly have that $\Sigma \models E^2(C)\Sigma' \wedge \Sigma'$. Moreover, $K(C)\Sigma = K(C)\Sigma'\Sigma$. Hence, the two bullets of the definition of solutions is trivially satisfied by that fact that $(\Sigma, \sigma) \in \text{Sol}(C)$. Therefore, we conclude that $(\Sigma, \sigma) \in \text{Sol}(C')$.

Let us now consider the case where our assumption do no hold. Thus since we assume that at least one rule is applicable on C , there exists $X \vdash^? u \in D(C)$ where $u \notin \mathcal{X}^1$. Let us do a case analysis on $X\Sigma$ since $X\Sigma \in \text{Conseq}(K\Sigma\sigma)$ by definition of a solution.

- either $X\Sigma \in \mathcal{F}_0$: in such a case, we have $C \rightarrow C'$ by Rule (MGS-CONSEQ) and we can prove similarly as in the previous case that $(\Sigma, \sigma) \in \text{Sol}(C')$;
- or $X\Sigma = f(\xi_1, \dots, \xi_n)$ where $\xi_i \in \text{Conseq}(K\Sigma\sigma)$ for all i : note that we know that $X\Sigma\Phi\sigma \downarrow = u\sigma$. Hence $u = f(u_1, \dots, u_n)$ for some u_1, \dots, u_n . We deduce that for all i , $\Phi\sigma \models \xi_i \vdash^? u_i\sigma$. Thus, by considering $\Sigma' = \{X_i \mapsto \xi_i\}_{i=1}^n$, we can conclude that $C \rightarrow C'$ by Rule (MGS-CONS) and $(\Sigma\Sigma', \sigma) \in \text{Sol}(C')$;
- or $X\Sigma \vdash^? u\sigma \in K\Sigma\sigma$ (since once again $X\Sigma\Phi\sigma \downarrow = u\sigma$): thus there exists $\xi \vdash^? v \in K$ such that $\xi\Sigma = X\Sigma$ and $u\sigma = v\sigma$. Hence $\text{mgu}\xi X$ exists and $\sigma \models u =^? v$. Therefore, we can conclude that $C \rightarrow C'$ by Rule (MGS-RES) and $(\Sigma, \sigma) \in \text{Sol}(C')$. ■

LEMMA B.17. *Let $C \neq \perp$ an extended constraint system such that $C \xi = C$, $\text{Inv}_{wf}(C)$ and $\text{Inv}_{sound}(C)$. If $C \not\rightarrow_{\xi}$ and C is a solved extended constraint system then $\text{mgs}(C) = \{\text{mgu}(E^2)\}$.*

PROOF. We know that for all $(\Sigma, \sigma) \in \text{Sol}(C)$, $\Sigma \models E^2(C)$ thus we directly obtain the existence of Σ' such that $\Sigma = \text{mgu}(E^2)\Sigma'$. Consider now the second bullet point of the definition of most general solutions. We know that C is solved. Hence consider a fresh bijective renaming Σ_1 from $\text{vars}^2(\Sigma_0) \cup \text{vars}^2(C) \setminus \text{dom}(\Sigma_0)$ to \mathcal{F}_0 . Let us define $\sigma_1 = \{x \mapsto X\Sigma_1 \mid X \vdash^? xD(C)\}$. Thanks to $\text{Inv}_{wf}(C)$, $\text{Inv}_{sound}(C)$ and Lemma B.14, B.12, and B.15 that $(\Phi\text{mgu}(E^1(C))\sigma_1, \text{mgu}(E^2)\Sigma_1, \text{mgu}(E^1(C))\sigma_1) \models D \wedge E^1 \wedge E^2$. Moreover, by Lemma B.14, we know that the first bullet of the definition of solution is satisfied. Finally, the second bullet is sat-

isfied otherwise Rule (MGS-UNSAT) would be applicable which contradict $C \not\approx \xi = C$. Therefore, $(mgu(E^2)\Sigma_1, mgu(E^1(C))\sigma_1) \in \text{Sol}(C)$. We conclude that $\text{mgs}(C) = \{mgu(E^2)\}$. ■

LEMMA B.18. *Let C an extended constraint system such that $C \not\approx \xi = C$, $\text{Inv}_{wf}(C)$ and $\text{Inv}_{sound}(C)$. If $C \not\approx \xi$ and C is not solved then $\text{Sol}(C) = \emptyset$.*

PROOF. Since C is not solved, we have two possibilities: Either (a) all deduction facts in D are have variables as right hand term but not pairwise distinct. But in such a case Rule (MGS-CONSEQ) would be applicable which contradicts $C \not\approx \xi$; or (b) there exists $(X \vdash^? u) \in D(C)$ such that $u \notin \mathcal{X}^1$. Since Rule (MGS-CONSEQ) is not applicable, we deduce that $u \notin \mathcal{F}_0$ and for all $\xi, \zeta \in R(C) \setminus \{X\}$, $(\xi, u) \notin \text{Conseq}(K \cup D)$. But rule Rule (MGS-CONS) is not applicable therefore, we deduce that $u \in \mathcal{N}$.

Assume now that $\text{Sol}(C) \neq \emptyset$ and so $(\Sigma, \sigma) \in \text{Sol}(C)$. Thus $X\Sigma\Phi \downarrow = u$. By definition of a solution, we know that $(X\Sigma, u) \in \text{Conseq}(K(C)\Sigma\sigma)$. Since $u \in \mathcal{N}$ it implies that there exists $(\xi \vdash^? v) \in K(C)$ such that $X\Sigma = \xi\Sigma$ and $u = v\sigma$. Note that by $\text{Inv}_{wf}(C)$, we also have that $v \notin \mathcal{X}^1$ and so $u = v$. In such a case, we obtain a contradiction with the fact the Rule (MGS-RES) is not applicable. ■

B.5 Correctness of the partition tree

In this section we prove the correctness of the procedure generating the partition tree, using the invariants proved in Appendix B.1. Let us first start by noticing that the case distinction rules and simplification rules preserves the first order solutions of the extended constraint systems. This property is stated in the following lemma.

LEMMA B.19. *Let \mathbb{S} be a set of set of extended symbolic processes such that $\text{Inv}_{all}(\mathbb{S})$. Let $\mathbb{S} \rightarrow \mathbb{S}'$ by applying only case distinction or simplifications rules (i.e. no symbolic transitions). Then:*

— Soundness:

for all $S \in \mathbb{S}$, for all $(\mathcal{P}, C, C^e) \in S$, for all $(\Sigma, \sigma) \in \text{Sol}(C^e)$, there exist $S' \in \mathbb{S}'$, $(\mathcal{P}, C, C^e) \in S'$ and $(\Sigma', \sigma') \in \text{Sol}(C^e)$ such that $\sigma|_{\text{vars}^1(C)} = \sigma'|_{\text{vars}^1(C)}$

— Completeness:

for all $S' \in \mathbb{S}'$, for all $(\mathcal{P}, C, C^e) \in S'$, for all $(\Sigma', \sigma') \in \text{Sol}(C^e)$, there exist $S \in \mathbb{S}$, $(\mathcal{P}, C, C^e) \in S$ and $(\Sigma, \sigma) \in \text{Sol}(C^e)$ such that $\Sigma|_{\text{vars}^2(C)} = \Sigma'|_{\text{vars}^2(C)}$ and $\sigma|_{\text{vars}^1(C)} = \sigma'|_{\text{vars}^1(C)}$

PROOF. We do a case analysis on the rule applied.

► *case 1:* Normalisation rules (simplification rules of Figure 8)

First, let us notice the result directly hold for Rules NORM-UNIF, NORM-FORMULA, and NORM-DUPL. Indeed, Rule NORM-DISEQ does not modify constraints on recipe and preserves the constraints on protocol terms. Moreover, Rule NORM-FORMULA, NORM-DUPL affect F which

do not impact the solutions of the extended constraint system. For Rule NORM-NO-MGS, since $mgs(C^e) = \emptyset$, we have by definition of most general unifiers that $Sol(C^e) = \emptyset$ (otherwise the first bullet of the definition is contradicted). Hence the result holds since $Sol(\perp) = \emptyset$. Similarly, Rule NORM-FORMULA checks whether the disequations $\forall \tilde{x}.\phi$ is trivially true meaning that the rule preserves the solutions.

► *case 2: Simplification rules on partitions of extended symbolic processes (Figure 9)*

The rule VECT-RM-UNSAT only removes an extended symbolic process with an extended constraint systems having no solution hence the result holds. Rule VECT-SPLIT splits a set of \mathbb{S} into two sets thus preserving the extended symbolic processes, and Rule VECT-ADD-FORMULA only adds element in F which do not impact the solutions of a constraint system. Therefore, for all these rules, the result hold. For Rule VECT-ADD-CONSEQ however, the result is not direct since the rule adds an element in the set K which has an impact on the solutions of a constraint system. However, we know from the application condition of the rule that the head protocol terms of the deduction facts added in K_i are not consequence of $K_i \cup D_i$. But we also know that $Inv_{satur}(\mathbb{S})$ and $Inv_{wf}(\mathbb{S})$ hold hence it implies that the recipe ξ (see Figure 9) contains $ax_{|\Phi_i|}$ and $vars^2(D_i) \cap \mathcal{X}_{\leq |\Phi_i|}^2 = \emptyset$. Hence, ξ cannot appear in the second order solutions C_i^e which allows us to conclude that the solutions are preserved.

► *case 3: Case distinction rules*

The case of case distinction rules is straightforward. Indeed, by definition all rules (SAT), (EQ) and (REW) always refine a set of extended symbolic process Γ into Γ_1, Γ_2 where Γ_1 is obtained by applying a substitution Σ to Γ , and Γ_2 by adding the constraint $\neg\Sigma$ to Γ . In particular this refinement preserves the solutions as expected. ■

Now we can show that the static equivalence is preserved by application of the case distinction and simplification rules.

LEMMA B.20. *Let \mathbb{S} be a set of set of extended symbolic processes such that $Inv_{all}(\mathbb{S})$. Let $\mathbb{S} \rightarrow \mathbb{S}'$ by applying only case distinction or simplifications rules (i.e. no symbolic transitions), $\Gamma \in \mathbb{S}$, $(\mathcal{P}_1, C_1, C_1^e), (\mathcal{P}_2, C_2, C_2^e) \in \Gamma$, $(\Sigma, \sigma_1) \in Sol(C_1^e)$ and $(\Sigma, \sigma_2) \in Sol(C_2^e)$ such that $\Phi(C_1)\sigma_1 \sim \Phi(C_2)\sigma_2$.*

PROOF. Once again, let us consider the potential rule applied.

► *case 1: Simplification rules*

The only non trivial case is Rule VECT-SPLIT (the other ones do not refine the partition and the conclusion is therefore immediate). However by Lemma applied to \mathbb{S} we know that if a deduction fact occurs in constraint systems C_1^e but no recipe equivalent formula can be found in the constraint system C_2^e , then no solution of C_2^e can satisfy the head of the formula.

Besides by $\text{Inv}_{\text{sound}}(\mathbb{S})$ we also know that all solutions of C_1^e satisfy this deduction fact. Then since $(\Sigma, \sigma_1) \in \text{Sol}(C_1^e)$, $(\Sigma, \sigma_2) \in \text{Sol}(C_2^e)$ and $\Phi(C_1)\sigma_1 \sim \Phi(C_2)\sigma_2$, we obtain a contradiction. Therefore, C_1^e and C_2^e are necessarily in the same set of \mathbb{S}' .

► *case 2: Case distinction rules*

Note that for case distinction rules, the proof is simple since each rule create a partition of the second-order solutions with respect to some mgs Σ_0 . Thus, assume w.l.o.g. that $(\Sigma', \sigma'_1) \in \text{Sol}(C_1^{e'})$.

► *case 2a: negative branch of the rule*

First consider that S' corresponds to branch in which we applied $\neg\Sigma_0$. In such a case, since we already know that Σ' satisfies $\neg\Sigma_0$ and no other constraint is added, we directly obtain from $(\Sigma, \sigma_2) \in \text{Sol}(C_2^e)$ that $(\Sigma', \sigma'_2) \in \text{Sol}(C_2^{e'})$ (in this case, we even have $\sigma_2 = \sigma'_2$).

► *case 2b: positive branch of the rule*

Now consider that S' corresponds to the branch in which we applied Σ_0 . In such a case, the application of Σ_0 on C_2^e regroups all the solution of C_2^e that satisfies Σ_0 . Since we know that $(\Sigma, \sigma_2) \in \text{Sol}(C_2^e)$ and $\Sigma'_{|\text{vars}^2(C_1)} = \Sigma_{|\text{vars}^2(C_1)}$ which implies $\Sigma'_{|\text{vars}^2(C_2)} = \Sigma_{|\text{vars}^2(C_2)}$, the result holds. ■

The previous two lemmas allow us to obtain the soundness and completeness properties of the partition tree. Note that the monoticity of the second-order predicate is also proved by B.19 (Completeness part) since a solution of a child constraint system is also a solution of parent one. We now need to prove that all nodes of the partition tree are valid configurations. For that we prove properties on extended constraint systems such that no more case distinction rules are applicable.

LEMMA B.21. *Let \mathbb{S} be a set of sets of extended symbolic processes such that $\text{Inv}_{\text{all}}(\mathbb{S})$ and no instance of the rule (SAT) or normalisation rules (i.e. the simplification rules of Figure 8) are applicable. For all $S \in \mathbb{S}$, for all $(\mathcal{P}, C, C^e) \in S$, writing $C^e = (\Phi, D, E^1, E^2, K, F)$ we have that*

1. C^e is solved
2. all formulas $\psi \in F$ are solved
3. E^1 does not contain disequations

PROOF. First of all the non-applicability of Rule (SAT) case 1 gives that either C^e is solved or $\text{mgs}(C^e) = \emptyset$; due to normalisation rules not being applicable we deduce that $\text{mgs}(C^e) \neq \emptyset$ meaning that C^e is solved. Similarly by the non applicability of case 2 we know that for all $\psi \in F$, either ψ is solved or $\text{mgs}(C^e[E^1 \mapsto E^1 \wedge \text{hyp}(\psi)]) = \emptyset$. But since the normalisation rules are also not applicable, we know that $\text{mgs}(C^e[E^1 \mapsto E^1 \wedge \text{hyp}(\psi)]) \neq \emptyset$: therefore ψ is solved. Finally the non applicability of case 3 and of the normalisation rules also gives us that E^1 is only composed of syntactic equations. ■

LEMMA B.22. *Let \mathbb{S} be a set of sets of extended symbolic processes such that $\text{Inv}_{\text{all}}(\mathbb{S})$ and no instance of the rule (SAT) or normalisation rules are applicable. For all $S \in \mathbb{S}$, for all $(\mathcal{P}, C, C^e) \in S$, $|\text{mgs}(C^e)| = 1$.*

PROOF. Let us denote $C^e = (\Phi, D, E^1, E^2, K, F)$. By Lemma B.21 we know that C^e is solved, that all formulas $\psi \in F$ are solved, and that E^1 only contain equations.

► *Step 1:* Construction of (Σ, σ) such that $(\Phi, \Sigma, \sigma) \models D \wedge E^1 \wedge E^2$

Since C^e is solved, we deduce that all deduction facts in $D = \{X_i \vdash^? x_i\}_{i=1}^n$ for some n and pairwise distinct x_i s and X_i s. Consider now the substitutions $\Sigma_0 = \{X_i \rightarrow n_i\}_{i=1}^n$ and $\sigma_0 = \{x_i \rightarrow n_i\}_{i=1}^n$ where the n_i s are pairwise distinct public names, i.e. $n_i \in \mathcal{F}_0$. Since no more normalisation rules are applicable, we know that the disequations in E^2 not trivially unsatisfiable. Therefore by replacing the free variables of the disequations by names allow us to obtain that Σ_0 the disequations of E^2 . By considering $\Sigma = \text{mgu}(E^2)\Sigma_0$, we obtain that $\Sigma \models E^2$. Moreover we proved that E^1 does not contain any disequations, we directly obtain that $\text{mgu}(E^1)\sigma_0 \models E^1$. Therefore, by defining $\sigma = \text{mgu}(E^1)\sigma_0$, we obtain that $(\Phi, \Sigma, \sigma) \models D \wedge E^1 \wedge E^2$.

► *Step 2:* Proof that (Σ, σ) is a solution

To prove that (Σ, σ) is an actual solution of C^e it remains to prove that it verifies the additional required two conditions: K-basis and uniformity. Let us first prove the K-basis, i.e. that for all $\xi \in \text{st}(\text{img}(\Sigma)) \cup \text{sst}^2(K\Sigma)$, $\text{msg}(\xi\Phi\sigma)$ and $(\xi, \xi\Phi\sigma) \in \text{Conseq}(K\Sigma\sigma)$. The case $\xi \in \text{sst}^2(K\Sigma)$ directly follows from $\text{Inv}_{\text{wf}}(C^e)$. Let us therefore consider the case $\xi \in \text{st}(\text{img}(\Sigma))$. Since $\text{Inv}_{\text{wf}}(C^e)$ holds we have that $\text{img}(\text{mgu}(E^2)) \subseteq \text{Conseq}(K \cup D)$; for the same reason we have that for all $\zeta \vdash^? u \in K$, $\text{st}(\zeta) \subseteq \text{Conseq}(K \cup D)$. Therefore by applying Lemma B.12, and by a quick induction on the size of the recipe in $\text{img}(\Sigma)$, we obtain that $\xi \in \text{Conseq}(K\Sigma)$. Finally, by definition of consequence and since $\text{Inv}_{\text{sound}}(C^e)$ holds, we have $\text{msg}(\xi\Phi\sigma)$ hence the K-basis.

Let us now prove uniformity. We know that C^e is solved which therefore means that for all recipes $\xi, \zeta \in \text{st}_c(\text{img}(\text{mgu}(E^2)), K \cup D)^2 \cup (\mathcal{F}_0 \times \text{vars}^2(D))$, $(\xi, u), (\zeta, u) \in \text{Conseq}(K \cup D)$ implies $\xi = \zeta$. Since $\Sigma = \text{mgu}(E^2)\Sigma_0$ we directly obtain that for all $\xi, \zeta \in \text{st}_c(\Sigma, K\Sigma)$, $(\xi, u), (\zeta, u) \in \text{Conseq}(K\Sigma)$ implies $\xi = \zeta$, which is exactly the uniformity.

► *Step 3:* Unicity of the solution

This step is rather straightforward: considering that $\Sigma = \text{mgu}(E^2)\Sigma_0$ and any other solutions $(\Sigma', \sigma') \in \text{Sol}(C^e)$ satisfy $\Sigma' \models E^2$, we deduce that $\text{mgs}(C^e) = \{\text{mgu}(E^2)\}$ and so $|\text{mgs}(C^e)| = 1$. ■

Let us now show that all extended constraint systems in the set have the same solutions and that they are statically equivalent.

LEMMA B.23. *Let \mathbb{S} be a set of set of extended symbolic processes such that $\text{Inv}_{\text{all}}(\mathbb{S})$ and no instances of the rules (SAT), (EQ) or (REW) or simplification rules are applicable. For all $S \in \mathbb{S}$, for all $(\mathcal{P}_1, C_1, C_1^e), (\mathcal{P}_2, C_2, C_2^e) \in S$, if $(\Sigma, \sigma_1) \in \text{Sol}(C_1^e)$ then $(\Sigma, \sigma_2) \in \text{Sol}(C_2^e)$ and $\Phi(C_1^e)\sigma_1 \sim \Phi(C_2^e)\sigma_2$.*

PROOF. Since (SAT) and normalisation rules are not applicable, we know by Lemma B.21 that all extended constraint systems $C^e \in S$ have a particular form, that is 1. all deduction facts in $D(C^e)$ have pairwise distinct variables as right hand side; and 2. $E^1(C^e)$ only contain syntactic equations. Moreover, we know that all extended constraint systems have the same structure. Therefore, if $(\Sigma, \sigma_1) \in Sol(C_1^e)$, we deduce that $\Sigma \models E^2(C_1^e)$ and for all $\xi \in st(img(\Sigma))$, $\xi \in Conseq(K(C_1^e)\Sigma)$, meaning that $\Sigma \models E^2(C_2^e)$ and $\xi \in Conseq(K(C_2^e)\Sigma)$. Since the first order solutions are always completely defined by the second-order substitutions, we can build σ_2' such that for all $X \vdash^? x \in D(C_2^e)$, $X\Sigma(\Phi(C_2^e)\sigma_2') \downarrow = x\sigma_2'$. Moreover, since $Inv_{sound}(C_1^e)$ and $Inv_{sound}(C_2^e)$ both hold and since for all $\xi \in st(img(\Sigma))$, $\xi \in Conseq(K(C_2^e)\Sigma)$, we deduce that for all $\xi \in st(img(\Sigma))$, $msg(\xi\Phi(C_2^e)\sigma_2')$. Note that we also need to satisfy the syntactic equations in E^1 . However thanks to $Inv_{wf}(C_2^e)$ holding, we know that $dom(mgu(E^1(C_2^e)) \cap vars^1(D(C_2^e))) = \emptyset$. Thus, we can build $\sigma_2 = mgu(E^1(C_2^e))\sigma_2'$ and obtain that $(\Sigma, \sigma_2) \models D(C_2^e) \wedge E^1(C_2^e) \wedge E^2(C_2^e)$. Note that by origination property of an extended constraint system, we have $\Phi(C_2^e)\sigma_2' = \Phi(C_2^e)\sigma_2$. Therefore, since we already prove that for all $\xi \in st(img(\Sigma))$, $msg(\xi\Phi(C_2^e)\sigma_2')$ and $\xi \in Conseq(K(C_2^e)\Sigma)$, it only remains to prove the second bullet point of Definition the definition of solutions extended constraint system to obtain that $(\Sigma, \sigma_2) \in Sol(C_2^e)$.

To prove this it suffices to prove that $\Phi(C_1^e)\sigma_1 \sim \Phi(C_2^e)\sigma_2$: the conclusion will then follow since $(\Sigma, \sigma_1) \in Sol(C_1^e)$. Therefore we let recipes ξ, ξ' and show that:

- (i) $msg(\xi\Phi(C_1^e)\sigma_1)$ iff $msg(\xi\Phi(C_2^e)\sigma_2)$
- (ii) if $msg(\xi\Phi(C_1^e)\sigma_1), \xi'\Phi(C_1^e)\sigma_1$ then $\xi\Phi(C_1^e)\sigma_1 \downarrow = \xi'\Phi(C_1^e)\sigma_1 \downarrow$ iff $\xi\Phi(C_2^e)\sigma_2 \downarrow = \xi'\Phi(C_2^e)\sigma_2 \downarrow$.

We prove this by lexicographic induction on $(N(\xi, \xi'), \max(|\xi|, |\xi'|))$ where $N(\xi, \xi')$ is the number of subterms $\zeta \in st(\xi, \xi')$ such that $\zeta \notin Conseq(K(C_1^e)\Sigma)$ (recall that since C_1^e and C_2^e have the same structure, we have $\zeta \in Conseq(K(C_1^e)\Sigma)$ iff $\zeta \in Conseq(K(C_2^e)\Sigma)$).

- *case 1:* $N(\xi, \xi') = 0$ and $\max(|\xi|, |\xi'|) = 0$

Impossible since there exist no terms of size 0.

- *case 2:* $N(\xi, \xi') > 0$

- *subgoal 2a:* Proof of (i)

Assume $msg(\xi\Phi(C_1^e)\sigma_1)$. Let us also assume by contradiction that $\neg msg(\xi\Phi(C_2^e)\sigma_2)$. Since we know that $N(\xi, \xi') > 0$, there exists $\zeta \in st(\xi, \xi')$ such that $\zeta \notin Conseq(K(C_1^e))$. Without loss of generality we can consider that $\zeta \in st(\xi)$ (otherwise we can apply our inductive hypothesis on ξ twice since $N(\xi, \xi')$ would be equal to 0 and so we would obtain a contradiction). Moreover, let us consider ζ such that $|\zeta|$ is minimal. Therefore, by definition of consequence, we deduce that $\zeta = g(\zeta_1, \dots, \zeta_n)$ with $g \in \mathcal{F}_d$ and for all $i \in \{1, \dots, n\}$, $\zeta_i \in Conseq(K(C_1^e))$. Since $msg(\xi\Phi(C_1^e)\sigma_1)$ we also deduce that $g(\zeta_1, \dots, \zeta_n)\Phi(C_1^e)\sigma_1 \downarrow$ is a protocol term. Therefore, there exist a rewrite rule $g(\ell_1, \dots, \ell_n) \rightarrow r$ and a substitution γ such that $\ell_i\gamma = \zeta_i\Phi(C_1^e)\sigma_1 \downarrow$ for all $i = 1 \dots n$.

Recall that the rule (REW) is not applicable on C_1^e and C_2^e . Therefore we can show that provided $\neg \text{msg}(\xi\Phi(C_2^e)\sigma_2)$ and $\text{g}(\zeta_1, \dots, \zeta_n)\Phi(C_1^e)\sigma_1 \downarrow$ is a protocol term then we necessarily have that there exists $\zeta'_1, \dots, \zeta'_n$ and u such that $\text{g}(\zeta'_1, \dots, \zeta'_n) \vdash^? u_1 \in F(C_1^e)$ and $\zeta'_i \Sigma \Phi(C_1^e)\sigma_1 \downarrow = \zeta_i \Phi(C_1^e)\sigma_1 \downarrow$. Moreover, since the normalisation rules are also not applicable (in particular Rule VECT-SPLIT), we deduce that there exists u_2 such that $\text{g}(\zeta'_1, \dots, \zeta'_n) \vdash^? u_2 \in F(C_2^e)$. By $\text{Inv}_{wf}(C_1^e)$, we know that for all $i \in \{1, \dots, n\}$, $\zeta'_i \in \text{Conseq}(\text{K}(C_1^e) \cup \text{D}(C_1^e))$ and so $\zeta'_i \Sigma \in \text{Conseq}(\text{K}(C_1^e)\Sigma)$. Moreover, by hypothesis on ζ_i , we know that $\zeta_i \in \text{Conseq}(\text{K}(C_1^e)\Sigma)$. Thus, by applying our inductive hypothesis, we obtain that $\zeta_i \Phi(C_2^e)\sigma_2 \downarrow = \zeta'_i \Sigma \Phi(C_2^e)\sigma_2 \downarrow$. Moreover, by $\text{Inv}_{\text{sound}}(C_2^e)$, we know that $\text{g}(\zeta'_1, \dots, \zeta'_n)\Sigma\Phi(C_2^e)\sigma_2 \downarrow = u_2\sigma_2$ which is a protocol term. We conclude that $\text{g}(\zeta_1, \dots, \zeta_n)\Sigma\Phi(C_2^e)\sigma_2 \downarrow$ is a protocol term and thus $\text{msg}(\xi\Phi(C_2^e)\sigma_2)$ gives us a contradiction.

► *subgoal 2b*: Proof of (ii)

Assume now that $\xi\Phi(C_1^e)\sigma_1 \downarrow = \xi'\Phi(C_1^e)\sigma_1 \downarrow$, $\text{msg}(\xi\Phi(C_1^e)\sigma_1)$ and $\text{msg}(\xi'\Phi(C_1^e)\sigma_1)$. Let us once again take the smallest $\zeta \in \text{st}(\xi, \xi')$ such that $\zeta \notin \text{Conseq}(\text{K}(C_1^e))$. We already proved above that there exist $u_1, u_2, \text{g}, \zeta'_1, \dots, \zeta'_n, \zeta_1, \dots, \zeta_n$ such that:

- $\zeta = \text{g}(\zeta_1, \dots, \zeta_n)$
- $\text{g}(\zeta'_1, \dots, \zeta'_n) \vdash^? u_1 \in F(C_1^e)$
- $\text{g}(\zeta'_1, \dots, \zeta'_n) \vdash^? u_2 \in F(C_2^e)$
- for all $i \in \{1, \dots, n\}$, $\zeta_i \Phi(C_2^e)\sigma_2 \downarrow = \zeta'_i \Sigma \Phi(C_2^e)\sigma_2 \downarrow$ and $\zeta_i \Phi(C_1^e)\sigma_1 \downarrow = \zeta'_i \Sigma \Phi(C_1^e)\sigma_1 \downarrow$.

By $\text{Inv}_{\text{satur}}(C_1^e)$, we know that there exists β such that $(\beta, u_1) \in \text{Conseq}(\text{K}(C_1^e) \cup \text{D}(C_1^e))$. However the normalisation Rule VECT-ADD-FORMULA is not applicable on the set of extended symbolic processes. Thus, we deduce that there exists β' such that $(\beta', u_1) \in \text{Conseq}(\text{K}(C_1^e) \cup \text{D}(C_1^e))$ and $\text{g}(\zeta'_1, \dots, \zeta'_n) \stackrel{?}{=} \beta' \in F(C_1^e)$. Once again due to the normalisation Rule VECT-SPLIT, we obtain that $\text{g}(\zeta'_1, \dots, \zeta'_n) \stackrel{?}{=} \beta' \in F(C_2^e)$. But $\text{Inv}_{\text{sound}}(C_2^e)$ and $\text{Inv}_{\text{sound}}(C_1^e)$ hold meaning that $(\Phi(C_2^e)\sigma_2, \Sigma, \sigma_2) \models \text{g}(\zeta'_1, \dots, \zeta'_n) \stackrel{?}{=} \beta'$ and $(\Phi(C_1^e)\sigma_1, \Sigma, \sigma_1) \models \text{g}(\zeta'_1, \dots, \zeta'_n) \stackrel{?}{=} \beta'$.

Note that if p is the position of ζ in ξ then we have $N(\xi[\beta'\Sigma]_p, \xi') < N(\xi, \xi')$. Thus by applying our inductive hypothesis, we obtain that $(\Phi(C_2^e)\sigma_2, \Sigma, \sigma_2) \models \xi[\beta'\Sigma]_p \stackrel{?}{=} \xi'$. Since $(\Phi(C_2^e)\sigma_2, \Sigma, \sigma_2) \models \text{g}(\zeta'_1, \dots, \zeta'_n) \stackrel{?}{=} \beta'$ and $(\Phi(C_2^e)\sigma_2, \Sigma, \sigma_2) \models \text{g}(\zeta'_1, \dots, \zeta'_n) \stackrel{?}{=} \text{g}(\zeta_1, \dots, \zeta_n)$, we conclude that $(\Phi(C_2^e)\sigma_2, \Sigma, \sigma_2) \models \xi \stackrel{?}{=} \xi'$.

► *case 3*: $N(\xi, \xi') = 0$ and $\max(|\xi|, |\xi'|) > 0$

In such a case, we know that $\xi, \xi' \in \text{Conseq}(\text{K}(C_1^e)\Sigma)$ and $\xi, \xi' \in \text{Conseq}(\text{K}(C_2^e)\Sigma)$. By definition of consequence and by $\text{Inv}_{\text{sound}}(C_1^e)$ and $\text{Inv}_{\text{sound}}(C_2^e)$, we directly obtain that $\text{msg}(\xi\Phi(C_1^e)\sigma_1)$ and $\text{msg}(\xi\Phi(C_2^e)\sigma_2)$ (same thing for ξ'). Now assume that $(\Phi(C_1^e)\sigma_1, \Sigma, \sigma_1) \models \xi \stackrel{?}{=} \xi'$. Since both ξ, ξ' are consequences of $\text{K}(C_1^e)\Sigma$, we deduce that:

- either $\xi = f(\xi_1, \dots, \xi_n)$ and $\xi' = f(\xi'_1, \dots, \xi'_n)$ with $f \in \mathcal{F}_c$ and $(\Phi(C_1^e)\sigma_1, \Sigma, \sigma_1) \models \xi_i \stackrel{?}{=} \xi'_i$ for all i . Therefore, we can apply our inductive hypothesis on the (ξ_i, ξ'_i) s to conclude.

- or $\xi\Sigma, \xi'\Sigma \in K(C_1^e)\Sigma$: Since we know that the rule (EQ) is not applicable, it implies that $\xi =_f^? \xi' \in F(C_1^e)$ and so $\xi =_f^? \xi' \in F(C_2^e)$ thanks to the normalisation Rule VECT-SPLIT. Since $\text{Inv}_{\text{sound}}(C_2^e)$ holds, we can conclude that $(\Phi(C_2^e)\sigma_2, \Sigma, \sigma_2) \models \xi =_f^? \xi'$.
- or $\xi\Sigma \in K(C_1^e)\Sigma$ and $\xi' = f(\xi'_1, \dots, \xi'_n)$ with $f \in \mathcal{F}_c$; Once again since the rule (EQ) is not applicable, we deduce that there exists $\zeta'_1, \dots, \zeta'_n$ such that $\xi =_f^? f(\zeta'_1, \dots, \zeta'_n) \in F(C_1^e)$. Note from $\text{Inv}_{\text{sound}}(C_1^e)$ that in such a case, $(\Phi(C_1^e)\sigma_1, \Sigma, \sigma_1) \models \xi =_f^? f(\zeta'_1, \dots, \zeta'_n)$ meaning that $(\Phi(C_1^e)\sigma_1, \Sigma, \sigma_1) \models \xi'_i =_f^? \zeta'_i$ for all $i \in \{1, \dots, n\}$. Since $|\xi'_i \Phi(C_1^e)\sigma_1 \downarrow| < |\xi \Phi(C_1^e)\sigma_1 \downarrow|$, we can apply our inductive hypothesis on all (ξ'_i, ζ'_i) meaning that $(\Phi(C_2^e)\sigma_2, \Sigma, \sigma_2) \models f(\zeta'_1, \dots, \zeta'_n) =_f^? \xi'$. However, by the rule VECT-SPLIT not being applicable, $\xi =_f^? f(\zeta_1, \dots, \zeta_n) \in F(C_1^e)$ implies $\xi =_f^? f(\zeta'_1, \dots, \zeta'_n) \in F(C_2^e)$ and so by $\text{Inv}_{\text{sound}}(C_2^e)$, we obtain that $(\Phi(C_2^e)\sigma_2, \Sigma, \sigma_2) \models \xi =_f^? f(\zeta'_1, \dots, \zeta'_n)$ which allows us to conclude that $(\Phi(C_2^e)\sigma_2, \Sigma, \sigma_2) \models \xi =_f^? \xi'$. ■

C. Termination proof

C.1 For mgs

The termination of the computation of most general solutions mostly relied on the following result, yet to be proved:

PROPOSITION 5.3 (decrease of unused first-order terms during constraint solving). *Let C^e be an extended constraint system such that $C^e \xrightarrow{\zeta} C^e$ and the invariants $\text{Inv}_{\text{wf}}(C^e)$ and $\text{Inv}_{\text{sound}}(C^e)$ hold. Then let $C^e \xrightarrow{\Sigma} C^{e'} \neq \perp$. If this transition is derived with:*

1. Rule (MGS-CONSEQ): $|\text{unused}^1(C^{e'})| \leq |\text{unused}^1(C^e)|$
2. Rules (MGS-RES) or (MGS-CONS): $|\text{unused}^1(C^{e'})| < |\text{unused}^1(C^e)|$

PROOF. Consider first the simplification rule (MGS-UNIF) and the ones from Figure 7. They typically apply protocol term substitutions on the constraint system (they also effect recipe disequations that are irrelevant in $\text{unused}^1(C^e)$). Note that the applied substitution is always generated from terms already in the constraint system. As such $\mu^1(C^e \downarrow) = \mu^1(C^e)$ and so $\Phi(C^e \downarrow) \mu^1(C^e \downarrow) = \Phi(C^e) \mu^1(C^e)$, $K(C^e \downarrow) \mu^1(C^e \downarrow) = K(C^e) \mu^1(C^e)$ and $D(C^e \downarrow) \mu^1(C^e \downarrow) = D(C^e) \mu^1(C^e)$. Thus, we directly obtain that $|\text{unused}^1(C^e \downarrow)| \leq |\text{unused}^1(C^e)|$.

Let us look at Rules (MGS-CONSEQ), (MGS-CONS) and (MGS-RES) and let us consider $C^e \xrightarrow{\Sigma} C^{e'}$. The rule (MGS-CONSEQ) does not modify the protocol terms of the constraint systems by apply a recipe substitution. However, we show an invariant on the constraint systems that any $\xi, \zeta \in \text{st}_c(C^e)$ are consequence of $K \cup D$ as well as any of their subterms (see Definition B.1 in Appendix). Thus, we deduce from the definition of $\text{st}_c(C^e)$ that $\text{st}_c(C^e)\Sigma \subseteq \text{st}_c(C^{e'})$. To conclude that $|\text{unused}^1(C^{e'})| \leq |\text{unused}^1(C^e)|$, we rely on the technical Proposition B.11; in other words, if $(\xi, t) \in \text{Conseq}(K(C^e)\mu^1(C^e) \cup D(C^e)\mu^1(C^e))$ then $(\xi\Sigma, t) \in \text{Conseq}(K(C^{e'})\mu^1(C^{e'}) \cup D(C^{e'})\mu^1(C^{e'}))$. Since $\text{st}_c(C^e)\Sigma \subseteq \text{st}_c(C^{e'})$, we conclude that $|\text{unused}^1(C^{e'})| \leq |\text{unused}^1(C^e)|$.

Applying the same reasoning for the rule (MGS-RES), we can also show that $unused^1(C'^e) \leq unused^1(C^e)$. However, we can even show that this inequality is strict. Indeed, using the same the notation in the rule (MGS-RES), this rule is only applied if $C^e = C^e \downarrow$ and for all $\xi \in st_c(C^e) \setminus \{X\}$, $(\xi, u) \notin \text{Conseq}(K \cup D)$. Note that $C^e = C^e \downarrow$ implies that $K\mu^1 = K$ and $D\mu^1 = D$. Moreover, it also implies that $u \in unused^1 C^e$. However, in C'^e , we have that $(\xi, u\mu_1(C'^e)) \in \text{Conseq}(K(C'^e)\mu^1(C'^e) \cup D(C'^e)\mu^1(C'^e))$. Moreover, we show another invariant on the constraint system (see Definition B.1 in Appendix) that ensures us that $X \in st_c(C'^e)$ and so $\xi \in st_c(C'^e)$. Hence, we obtain that $u\mu^1(C'^e) \notin unused^1(C'^e)$ allowing us to conclude that $unused^1(C'^e) < unused^1(C^e)$. By applying the same reasoning, we can also show that $unused^1(C'^e) < unused^1(C^e)$ when the rule (MGS-CONS) is applied. ■

C.2 Exponential measure

Another argument left pending is that each component of the measure except the last one can be bounded by an exponential in the DAG size of the parameters of the problem. We give a bound for each of them, in particular relying on the bound on $unused^1$ proved in the body of the article.

1. $M_1(\Gamma) \leq |P, Q|_{\text{dag}}$:
by definition.

2. $M_2(\Gamma) \leq (|P|_{\text{dag}} |E|_{\text{dag}})^{|P|_{\text{dag}}} + (|Q|_{\text{dag}} |E|_{\text{dag}})^{|Q|_{\text{dag}}}$:

The measure corresponds to the number of symbolic transitions possible from P and Q for a given symbolic trace, hence the bound. Notice that the part $|E|_{\text{dag}}^{|P|_{\text{dag}}}$ is due to the computation of the most general unifiers modulo E in the symbolic transitions.

3. $M_3(\Gamma) \leq 9 |P, Q, E|_{\text{dag}}^3$:

It suffices to observe that $\text{set}_K(C^e) \leq unused^1(C^e)$ and to use the bound proved in the body of the article.

4. $M_4(\Gamma) \leq M_2(\Gamma)$:

Trivial.

5. $M_5(\Gamma) \leq M_2(\Gamma) \times |E|_{\text{dag}}^{|E|_{\text{dag}}} \times (18 |P, Q, E|_{\text{dag}})^{27|P, Q, E|_{\text{dag}}^3}$:

Bounding the size of $|\text{set}_{\text{REW}}(C^e)|$ can easily be done: the number of $\psi \in K$ possible is bounded by $|K|$, itself bounded by $|\text{set}_K(C^e)|$. The number of rewrite rules, position p and $\psi_0 \in \text{RewF}(\xi, \ell \rightarrow r, p)$ only depends on the rewrite systems and can be bounded by $|E|_{\text{dag}}^{|E|_{\text{dag}}}$. Note that the exponential comes mainly from the number of possible positions in ℓ . We already know that the number of most general solutions is bounded by $(|K(C^e)| + 1)^{unused^1(C^e)}$. Combining with all previous results, and with the rough approximation $9 |P, Q, E|_{\text{dag}}^3 + 1 \leq 18 |P, Q, E|_{\text{dag}}^3$, we obtain the above bound.

6. $M_6(\Gamma) \leq |E| \times M_2(\Gamma)$:

To bound this number, we need to recall that we always apply the case distinction rules with the priority ordering (SAT) < (REW). Thus, when we apply a rule (REW), there is no unsolved deduction formula in any of the extended constraint systems (otherwise we should have applied the rule (SAT)). It means this measure is bounded by the number of deduction formulas produced by one instance of (REW). By definition, we know that $|\text{RewF}(\xi, \ell \rightarrow r, p)| \leq |\mathcal{R}|$ (one formula per rewrite rule). Thus, the rule (REW) generates at most $|E| \times M_2(\Gamma)$ deduction formulas.

7. $M_7(\Gamma) \leq M_2(\Gamma) \times 2 |E|_{\text{dag}} (|P, Q|_{\text{dag}})^2 (1 + |E|_{\text{dag}})^2$:

The application conditions stipulate that the rule can be applied either (a) on two deduction facts of $K(C_i^e)$, or (b) on one deduction fact of $K(C_i^e)$ in combination with a construction function symbol.

Note that even though the rule also consider the existence of a most general solution $\Sigma \in \text{mgs}(C_i^e[E^1 \mapsto E^1 \wedge \text{hyp}(\psi:(\Sigma_0, C_i^e))])$, the number of applications of the rule (EQ) will not depend on the number of possible most general solutions. Indeed, consider the case (a) where the rule is applied on two deduction fact $(\xi_1 \vdash^? u_1), (\xi_2 \vdash^? u_2) \in K(C_i^e)$. Thus, an equality formula with $\xi_1 \Sigma =_f^? \xi_2 \Sigma$ as head will be added in $F(C_i^e:\Sigma)$. However, in the application conditions of the rule, we also require that *for all* $(\forall S. H \Leftarrow \varphi) \in F(C_i^e)$, $H \neq (\xi_1 =_f^? \xi_2)$. Thus, a new application of the rule (EQ) on $C_i^e:\Sigma$ with the same (up to instantiation of Σ) deductions facts from $K(C_i^e:\Sigma)$ will be prevented.

The same situation occurs in case (b) with the condition *for all* $(\forall S. \zeta_1 =_f^? \zeta_2 \Leftarrow \varphi) \in F(C_i^e)$, $\zeta_1 = \xi_1$ implies $\text{root}(\zeta_2) \neq f$. We therefore conclude that the rule (EQ) can be applied only once per pair of deduction facts in K and once per deduction fact in K and function symbol in \mathcal{F}_c .

8. $M_8(\Gamma) \leq M_2(\Gamma)$:

Unsolved equality formulas can be generated by two rules: the case distinction rule (EQ) or the simplification Rule VECT-ADD-FORMULA. However, once again because of the priority order (SAT) < (EQ), the two rules cannot be triggered simultaneously and the rule (EQ) is only triggered when there is no unsolved equality formulas. Note that due to the condition $\forall i. \forall (\forall S. \zeta_1 =_f^? \zeta_2 \Leftarrow \varphi) \in F_i, \zeta_1 \neq \xi$ or $\zeta_2 \neq \xi$ in Rule VECT-ADD-FORMULA, two instances of the Rule VECT-ADD-FORMULA with different recipes ζ (e.g. if u_1 can be deducible with two different recipes) cannot be applied sequentially. Thus, at any given moment, there is at most one unsolved equality formula per extended constraint system of Γ , hence the bound.

C.3 Bounding the increase of second order terms

In Section 5.3, Proposition 5.8, we gave a bound on the increase of the size of most general solutions when applying the rule (SAT). We give here the arguments to extend to the other

case distinction rules. For that it suffices to generalise this property to a more general set of substitution Σ :

DEFINITION C.1. Let C^e be an extended constraint system. Let Σ be a second-order substitution. We say that $\Sigma \in \text{CompatSubs}(C^e)$ if $\text{dom}(\Sigma) \subseteq \text{vars}^2(D(C^e))$ and for all $X \in \text{dom}(\Sigma)$, $X\Sigma \in \text{Conseq}(K(C^e) \cup D' \cup D_\Sigma)$ where $D' = \{X \vdash^? u \in D(C^e) \mid X \notin \text{dom}(\Sigma)\}$ and $D_\Sigma = \{X \vdash^? x \mid x \text{ fresh and } X \in \text{vars}^2(\Sigma) \setminus \text{vars}^2(C^e)\}$.

Intuitively, $\text{CompatSubs}(C^e)$ represents the recipe substitutions Σ that can be applied to the constraint system C^e , i.e. $C^e:\Sigma$, and such that the recipes in the of Σ would be consequence of $C^e:\Sigma$. Note that $\text{mgs}(C^e) \subseteq \text{CompatSubs}(C^e)$.

By applying Proposition B.11, we can show that:

$$\text{for all } \Sigma \in \text{CompatSubs}(C^e), |\text{unused}^1(C^e:\Sigma)| \leq |\text{unused}^1(C^e)| \quad (4)$$

Note that in a set of symbolic processes two extended constraint systems C_1^e, C_2^e always have the same *recipe structure* (Invariant Inv_{str}), i.e. $|\Phi(C_1^e)| = |\Phi(C_2^e)|$, $\text{vars}^2(C_1^e) = \text{vars}^2(C_2^e)$ and $\{\xi \mid (\xi \vdash^? u) \in K(C_1^e)\} = \{\xi \mid (\xi \vdash^? u) \in K(C_2^e)\}$. Thus, we deduce that $\text{CompatSubs}(C_1^e) = \text{CompatSubs}(C_2^e)$. Therefore, we can conclude that for any simplification and case distinction rules, $|\text{unused}^1(C^e)|$ never increase for all extended constraint systems in a set of extended symbolic processes.

D. Proofs of complexity lower bounds

D.1 Advanced winning strategies

Before starting the proofs, we present some characterizations of observational (in)equivalence in order to make the incoming proofs easier to handle.

REMARK D.1. The results of this section (D.1) also apply to the extended semantics of Section 5.5.1.

For the defender The transitions of the semantics which are deterministic and silent are not essential to equivalence proofs as they do not interfere substantially with them. We introduce below a refined proof technique to rule them out.

DEFINITION D.2 (simplification). A multiset of closed plain processes \mathbb{S} is silent in an extended process (\mathcal{P}, Φ) when for all transitions $(\mathcal{P} \cup \mathbb{S}, \Phi) \xrightarrow{\alpha} (Q, \Phi')$, it holds that $Q = \mathcal{P}' \cup \mathbb{S}$ with $(\mathcal{P}, \Phi) \xrightarrow{\alpha} (\mathcal{P}', \Phi')$ and \mathbb{S} silent in (\mathcal{P}', Φ') . Then we define \rightsquigarrow (simplification relation) the relation

on extended processes defined by the following inference rules:

$$\frac{\mathbb{S} \text{ silent in } (\mathcal{P}, \Phi)}{(\mathcal{P} \cup \mathbb{S}, \Phi) \rightsquigarrow (\mathcal{P}, \Phi)} [(S - sil)]$$

$$\frac{c \in \mathcal{N} \quad \text{msgt} \quad c \notin \text{names}_{\mathcal{P}, \Phi}}{(\mathcal{P} \cup \{\bar{c}\langle t \rangle.P, c(x).Q\}, \Phi) \rightsquigarrow (\mathcal{P} \cup \{P, Q\{x \mapsto t\}\}, \Phi)} [(S - comm)]$$

$$\frac{A \xrightarrow{\tau} B \text{ by rules NULL, PAR, THEN, ELSE}}{A \rightsquigarrow B} [(S - npte)]$$

In other words, we write $A \rightsquigarrow B$ when B is obtained from A by removing some silent process or applying a deterministic (in the sense of the confluence lemma below) instance of the transition relation $\xrightarrow{\tau}$. We call $\rightsquigarrow_{\text{pi}}$ the restriction of \rightsquigarrow to the rule simplification. Their reflexive transitive closures are denoted \rightsquigarrow^* and $\rightsquigarrow_{\text{pi}}^*$ respectively as usual.

LEMMA D.3. *If $A \rightsquigarrow B$ (by some rule ρ_{sil} of the definition of \rightsquigarrow) and $A \xrightarrow{\alpha} C$ (by some rule ρ_c of the semantics), then either $B = C$ and $\alpha = \rightsquigarrow^*$, or there exists D such that $C \rightsquigarrow D$ (by rule ρ_{sil}) and $B \xrightarrow{\alpha} D$ (by rule ρ_c).*

PROOF. We make a case analysis on the rule used to obtain the reduction $A \rightsquigarrow B$:

— **case 1** : by rule simplification:

Then we write $A = (\mathcal{P} \cup \mathbb{S}, \Phi)$, $B = (\mathcal{P}, \Phi)$. By definition of silent processes, the reduction $A \xrightarrow{\alpha} C$ hence gives $C = (\mathcal{P}' \cup \mathbb{S}, \Phi')$ where $(\mathcal{P}, \Phi) \xrightarrow{\alpha} (\mathcal{P}', \Phi') = D$ and \mathbb{S} silent in D . In particular D gives the expected conclusion.

— **case 2** : by rule simplification or simplification:

Then either $B = C$ and the conclusion is immediate, or $B \neq C$ and a quick analysis of the rules of the semantics gives $\mathcal{P}, Q_B, Q'_B, Q_C, Q'_C, \Phi, \Phi'$ such that:

$$A = (\{Q_B, Q_C\} \cup \mathcal{P}, \Phi) \quad B = (\{Q'_B, Q_C\} \cup \mathcal{P}, \Phi) \quad C = (\{Q_B, Q'_C\} \cup \mathcal{P}, \Phi')$$

$$(\{Q_B\}, \Phi) \rightsquigarrow (\{Q'_B\}, \Phi) \quad (\{Q_C\}, \Phi) \xrightarrow{\alpha} (\{Q'_C\}, \Phi')$$

and we conclude by choosing $D = (\{Q'_B, Q'_C\} \cup \mathcal{P}, \Phi')$. ■

COROLLARY D.4. *If $A \rightsquigarrow^* B$ and $A \xrightarrow{\alpha} C$ then either $B \rightsquigarrow^* C$ and $\alpha = \rightsquigarrow^*$, or there exists D such that $C \rightsquigarrow^* D$ and $B \xrightarrow{\alpha} D$.*

PROOF. By a straightforward induction on the number of steps of the reduction $A \rightsquigarrow^* B$. ■

COROLLARY D.5. $\rightsquigarrow_{\text{pi}}$ is convergent.

PROOF. The termination of $\rightsquigarrow_{\text{pi}}$ follows from the termination of the whole calculus. As for the local confluence (which suffices by Newmann's lemma), we observe that by Lemma D.3, if $A \rightsquigarrow_{\text{pi}} B$ and $A \rightsquigarrow_{\text{pi}} C$ then either $B = C$, or there is D such that $B \rightsquigarrow_{\text{pi}} D$ and $C \rightsquigarrow_{\text{pi}} D$: in particular, $B \rightsquigarrow_{\text{pi}}^* E$ and $C \rightsquigarrow_{\text{pi}}^* E$ for some $E \in \{C, D\}$. ■

In particular, all extended processes A have a unique normal form w.r.t. $\rightsquigarrow_{\text{pi}}$ which will be written $A_{\downarrow \text{pi}}$. This notation is lifted to multiset of processes, writing $\mathcal{P}_{\downarrow \text{pi}}$ (which is consistent since $\rightsquigarrow_{\text{pi}}$ does not modify the frame). With all of this, we eventually gathered all the ingredients to introduce our characterization of bisimilarity:

DEFINITION D.6 (bisimulation up to \rightsquigarrow). A symmetric relation \mathcal{R} on extended processes is then said to be bisimulation up to \rightsquigarrow , or a bisimulation up to simplification, when:

- $\mathcal{R} \subseteq \sim$;
- for all extended processes A, B such that $A \mathcal{R} B$, and for all transitions $A \xrightarrow{\alpha} A'$, there exists $B \xRightarrow{\alpha} B'$ such that $A' \rightsquigarrow \mathcal{R} \rightsquigarrow B'$.

PROPOSITION D.7. For all extended processes A and B , $A \approx_b B$ iff there exists a bisimulation up to simplification \mathcal{R} such that $A \rightsquigarrow \mathcal{R} \rightsquigarrow B$.

PROOF. The forward implication follows from the fact that \approx_b is a bisimulation up to simplification (by reflexivity of \rightsquigarrow^*). For the converse, let us consider \mathcal{R} a bisimulation up to \rightsquigarrow and prove that it is contained in \approx_b . In order to do that, it suffices to show that:

- $\rightsquigarrow \mathcal{R} \rightsquigarrow$ is symmetric;
- $(\rightsquigarrow \mathcal{R} \rightsquigarrow) \subseteq \sim$;
- for all extended processes A, B such that $A \rightsquigarrow \mathcal{R} \rightsquigarrow B$, if $A \xrightarrow{\alpha} A'$ then there exists $B \xRightarrow{\alpha} B'$ such that $A' \rightsquigarrow \mathcal{R} \rightsquigarrow B'$.

These three properties indeed justify that $(\rightsquigarrow \mathcal{R} \rightsquigarrow) \subseteq \approx_b$ by definition, hence the expected conclusion as $\mathcal{R} \subseteq \rightsquigarrow \mathcal{R} \rightsquigarrow$ by reflexivity of \rightsquigarrow^* . Yet it appears that the first two points directly follows from the properties of \mathcal{R} and the reflexivity of \rightsquigarrow^* , and we thus only need to prove the third point. Let us therefore consider the following hypotheses and notations:

$$A \rightsquigarrow^* C \quad \mathcal{R} \quad D \rightsquigarrow^* B \qquad A \xrightarrow{\alpha} A'$$

and let us exhibit B' such that $B \xRightarrow{\alpha} B'$ and $A' \rightsquigarrow \mathcal{R} \rightsquigarrow B'$. Let us consider the two cases induced by the application of Corollary D.4:

- **case 1 :** $A' \rightsquigarrow^* C$ and $\alpha = \rightsquigarrow^*$
Then we can choose $B' = B$.
- **case 2 :** there exists C' such that $A' \rightsquigarrow^* C'$ and $C \xrightarrow{\alpha} C'$
Consequently, since \mathcal{R} is a bisimulation up to simplification, there is D' such that $D \xRightarrow{\alpha} D'$ and $C' \rightsquigarrow \mathcal{R} \rightsquigarrow D'$. Then we remark that for all extended processes B_1, B_2, B_3 :

— a transition $B_1 \rightsquigarrow B_2$ with rules simplification or simplification implies $B_1 \xrightarrow{\tau} B_2$;
 — if $B_1 \rightsquigarrow B_2$ with rule simplification and $B_2 \xrightarrow{\beta} B_3$, then $B_1 \xrightarrow{\beta} \rightsquigarrow B_3$.
 In particular since $B \xrightarrow{\star} D \xRightarrow{\alpha} D'$, we have $B \xRightarrow{\alpha} D'' \xrightarrow{\star} D'$ for some D'' . Hence the conclusion by choosing $B' = D''$. ■

For the attacker When taking the negation of labelled bisimilarity, we essentially obtain a set of rules for a game whose states are pairs of processes (A, B) : an attacker selects a transition and a defender answers by selecting a equivalently-labelled sequence of transitions in the other process.

DEFINITION D.8 (labelled attack). A relation S on extended processes is called a labelled attack when for all A, B such that $A S B$, it holds that:

1. either: $A \not\sim B$
2. or: $\exists A \xrightarrow{\alpha \text{ tr}} A', \forall B \xRightarrow{\alpha \text{ tr}} B', A' S B'$
3. or: $\exists B \xrightarrow{\alpha \text{ tr}} B', \forall A \xRightarrow{\alpha \text{ tr}} A', A' S B'$

Note that labelled attacks are not the direct translation of the above intuition since they allow the attacker to choose several transitions in a row; this intuitively entails no loss of generality since it is equivalent to the attacker selecting some transitions non-adaptatively (i.e. independently of the answer of the defender). Here is the formal statement of correctness:

PROPOSITION D.9. For all extended processes A and B , $A \not\sim_b B$ iff there exists a labelled attack S such that $A S B$.

PROOF. The forward implication is immediate since $\not\sim_b$ is a labelled attack (we can even choose $\text{tr} = \varepsilon$ everytime). Let then S be a labelled attack such that $A S B$ and let us prove that $S \subseteq \not\sim_b$. More precisely, we prove that $S \subseteq S' \subseteq \not\sim_b$ for some relation S' . We will construct S' in such a way that for all A, B extended processes, $A S' B$ entails:

- (i) either: $A \not\sim B$;
- (ii) or: $\exists A \xrightarrow{\alpha} A', \forall B \xRightarrow{\alpha} B', A' S B'$;
- (iii) or: $\exists B \xrightarrow{\alpha} B', \forall A \xRightarrow{\alpha} A', A' S B'$

The inclusion $S' \subseteq \not\sim_b$ is indeed clear if this property is verified, hence the expected conclusion provided such a relation S' . We concretely define it as the smallest relation on extended

processes saturated by the following inference rules:

$$\frac{A \ S \ B}{A \ S' \ B} [(Axiom)]$$

$$\frac{A \ S' \ B \quad A \xrightarrow{\alpha} A' \xrightarrow{\alpha' \text{ tr}} A'' \quad \forall B \xrightarrow{\alpha.\alpha'.\text{tr}} B'', A'' \ S' \ B'' \quad B \xRightarrow{\alpha} B'}{A' \ S' \ B'} [(Dec - L)]$$

$$\frac{A \ S' \ B \quad B \xrightarrow{\alpha} B' \xrightarrow{\alpha' \text{ tr}} B'' \quad \forall A \xrightarrow{\alpha.\alpha'.\text{tr}} A'', A'' \ S' \ B'' \quad A \xRightarrow{\alpha} A'}{A' \ S' \ B'} [(Dec - R)]$$

In particular, note that $S \subseteq S'$ thanks to the rule For the attacker. As for the two other rules For the attacker and For the attacker, they intuitively decompose sequences $\xrightarrow{\alpha \text{ tr}}$ into atomic transitions in order to switch from points 2. or 3. of Definition D.8 to points (ii) or (iii).

Let then A and B be two extended processes such that $A \ S' \ B$. We consider a proof-tree of $A \ S' \ B$ in the inference system above and perform a case analysis on the rule at its root:

— **case 1** For the attacker: $A \ S \ B$.

As S is a labelled attack, we apply the case analysis of Definition D.8:

— **case 1.a** : $A \not\sim B$.

Then (i) is satisfied.

— **case 1.b** : there exists $A \xrightarrow{\alpha} A' \xRightarrow{\text{tr}} A''$ such that $A'' \ S \ B''$ for all $B \xRightarrow{\alpha.\text{tr}} B''$.

In particular, keeping in mind that $S \subseteq S'$ due to the rule For the attacker, we have $A'' \ S' \ B''$ for all $B \xRightarrow{\alpha.\text{tr}} B''$. Let us then show that the transition $A \xrightarrow{\alpha} A'$ satisfies (ii). We therefore have to show that $A' \ S' \ B'$ for all $B \xRightarrow{\alpha} B'$. If $A' = A''$ then the result follows from the hypothesis. Otherwise let us write $A \xrightarrow{\alpha} A' \xrightarrow{\alpha' \text{ tr}'} A''$ where $\alpha'.\text{tr}' = \text{tr}$ and the rule For the attacker justifies that $A' \ S' \ B'$ for all $B \xRightarrow{\alpha} B'$.

— **case 1.c** : there exists $B \xrightarrow{\alpha} B' \xRightarrow{\text{tr}} B''$ such that $B'' \ S \ A''$ for all $A \xRightarrow{\alpha.\text{tr}} A''$.

Analogous, targeting (iii) instead of (ii) and replacing For the attacker by For the attacker.

— **case 2** For the attacker: there are $A_0, A_1, A_2, B_0, B_2, \alpha, \alpha', \text{tr}$, such that $A_0 \ S' \ B_0, B_0 \xRightarrow{\alpha} B, A_0 \xrightarrow{\alpha} A \xrightarrow{\alpha'} A_1 \xRightarrow{\text{tr}} A_2$, and $\forall B_0 \xRightarrow{\alpha.\alpha'.\text{tr}} B_2, A_2 \ S' \ B_2$.

Let us show that the transition $A \xrightarrow{\alpha'} A_1$ satisfies (ii). We therefore have to show that $A_1 \ S' \ B_1$ for all $B \xRightarrow{\alpha'} B_1$. If $A_1 = A_2$ then the result follows from the hypothesis. Otherwise we write $A \xrightarrow{\alpha'} A_1 \xrightarrow{\alpha'' \text{ tr}'} A_2$ where $\alpha''.\text{tr}' = \text{tr}$ and the rule For the attacker justifies that $A_1 \ S' \ B_1$ for all $B \xRightarrow{\alpha'} B_1$.

— **case 3** For the attacker: Analogous to case 2. ■

D.2 Correctness of the encodings (Section 5.5.1)

Now we prove that the translation $\llbracket \cdot \rrbracket$ of the extended semantics is correct:

LEMMA D.10. *Let \approx_t^+ and \approx_b^+ be the notions of trace equivalence and labelled bisimilarity over the extended calculus (the flag $^+$ being omitted outside of this lemma). For all extended processes $A = (\mathcal{P}, \Phi)$, the translation $\llbracket A \rrbracket = (\llbracket \mathcal{P} \rrbracket, \Phi) = (\{\llbracket P \rrbracket \mid P \in \mathcal{P}\}, \Phi)$ can be computed in polynomial time, $A \approx_t^+ \llbracket A \rrbracket$ and $A \approx_b^+ \llbracket A \rrbracket$.*

But first of all, a (trivial) observation about the free variables of a translated process:

LEMMA D.11. *For all plain processes P and all first-order substitution σ , $\llbracket P\sigma \rrbracket = \llbracket P \rrbracket \sigma$.*

We will use this lemma implicitly in the remaining of this section. Besides, as we have the inclusion of relations $\approx_b \subseteq \approx_t$, we only need to prove the observational-equivalence statement of Lemma D.10. We recall that we use notations $A_{\downarrow \text{pi}}$ and $\mathcal{P}_{\downarrow \text{pi}}$ to refer to normal forms w.r.t. $\rightsquigarrow_{\text{pi}}$ (see Corollary D.5).

PROPOSITION D.12. *We consider \mathcal{R} the symmetric closure of:*

$$\{(C, \llbracket C \rrbracket_{\downarrow \text{pi}}) \mid C \text{ extended process such that } C = C_{\downarrow \text{pi}}\}$$

\mathcal{R} is a bisimulation up to simplification.

PROOF. \mathcal{R} is symmetric by definition and is trivially included in \sim . Let then $(A, B) \in \mathcal{R}$ and $A \xrightarrow{\alpha} A'$ and let us exhibit B' such that $B \xrightarrow{\alpha} B'$ and $A' \mathcal{R} B'$. We perform a case analysis on the rule triggering the transition $A \xrightarrow{\alpha} A'$:

— **case 1** (rules NULL, PAR, THEN, ELSE):

This case cannot arise as A is in normal form w.r.t. $\rightsquigarrow_{\text{pi}}$ by definition of \mathcal{R} .

— **case 2** (rule IN): $\alpha = \xi(\zeta)$ for some $\xi, \zeta \in \mathcal{T}(\mathcal{F}, \mathcal{F}_0 \cup \text{dom}\Phi)$ and:

$$\begin{array}{ll} A = (\mathcal{P} \cup \{u(x).P\}, \Phi) & \text{with msg}_u, \text{msg}_{\xi\Phi} \text{ and } \xi\Phi \downarrow = u \downarrow \\ A' = (\mathcal{P} \cup \{P\{x \mapsto \zeta\Phi\}\}, \Phi) & \text{with msg}_{\zeta\Phi} \end{array}$$

Then, by a case analysis on the hypothesis $A \mathcal{R} B$:

— **case 2.a** : $B = \llbracket A \rrbracket_{\downarrow \text{pi}}$

Then we can write:

$$B = (\llbracket \mathcal{P} \rrbracket_{\downarrow \text{pi}} \cup \{u(x). \llbracket P \rrbracket\}, \Phi)$$

and we conclude by remarking that $B \xrightarrow{\xi(\zeta)} B' = (\llbracket \mathcal{P} \rrbracket_{\downarrow \text{pi}} \cup \{\llbracket P \rrbracket\{x \mapsto \zeta\Phi\}\}, \Phi)$ and:

$$A' \overset{\star}{\rightsquigarrow} A'_{\downarrow \text{pi}} \mathcal{R} \llbracket A'_{\downarrow \text{pi}} \rrbracket_{\downarrow \text{pi}} = \llbracket A' \rrbracket_{\downarrow \text{pi}} = (\llbracket \mathcal{P} \rrbracket_{\downarrow \text{pi}} \cup \{\llbracket P \rrbracket\{x \mapsto \zeta\Phi\}\}_{\downarrow \text{pi}}, \Phi) \overset{\star}{\rightsquigarrow} B'$$

— **case 2.b** : $A = \llbracket B \rrbracket_{\downarrow \text{pi}}$ (and $B = B_{\downarrow \text{pi}}$)

Note that $u = \xi\Phi\downarrow$ cannot be one of the names introduced by the translation $\llbracket \cdot \rrbracket$: these names can indeed not appear in Φ since they are chosen private and fresh and since the semantics cannot introduce new private names. In particular we can write:

$$B = (Q \cup \{u(x).Q\}, \Phi) \quad \text{with } \llbracket Q \rrbracket = P \text{ and } \llbracket Q \rrbracket_{\downarrow \text{pi}} = \mathcal{P}$$

and we conclude by writing $B \xrightarrow{\xi(\zeta)} B' = (Q \cup \{Q\{x \mapsto \zeta\Phi\downarrow\}\}, \Phi)$ and:

$$A' \overset{\star}{\rightsquigarrow} (\llbracket Q \rrbracket_{\downarrow \text{pi}} \cup \{\llbracket Q \rrbracket\{x \mapsto \zeta\Phi\downarrow\}\}_{\downarrow \text{pi}}, \Phi) = \llbracket B' \rrbracket_{\downarrow \text{pi}} = \llbracket B'_{\downarrow \text{pi}} \rrbracket_{\downarrow \text{pi}} \mathcal{R} B'_{\downarrow \text{pi}} \overset{\star}{\leftarrow} B'$$

— **case 3** (rule OUT): $\alpha = \bar{\xi}\langle a x_n \rangle$ for some $\xi, \in \mathcal{T}(\mathcal{F}, \mathcal{F}_0 \cup \text{dom}\Phi)$, $a x_n \in \mathcal{AX}$ and:

$$\begin{aligned} A &= (\mathcal{P} \cup \{\bar{u}\langle t \rangle.P\}, \Phi) && \text{with } \text{msg}u, \text{msg}t, \text{msg}\xi\Phi \text{ and } \xi\Phi\downarrow = u\downarrow \\ A' &= (\mathcal{P} \cup \{P\}, \Phi') && \text{where } \Phi' = \Phi \cup \{a x_n \mapsto t\downarrow\} \text{ and } n = |\Phi| + 1 \end{aligned}$$

Then, by a case analysis on the hypothesis $A \mathcal{R} B$:

— **case 3.a** : $B = \llbracket A \rrbracket_{\downarrow \text{pi}}$

Then we can write:

$$B = (\llbracket \mathcal{P} \rrbracket_{\downarrow \text{pi}} \cup \{\bar{u}\langle t \rangle.\llbracket P \rrbracket\}, \Phi)$$

and we conclude by remarking that $B \xrightarrow{\bar{\xi}\langle a x_n \rangle} B' = (\llbracket \mathcal{P} \rrbracket_{\downarrow \text{pi}} \cup \{\llbracket P \rrbracket\}, \Phi)$ and:

$$A' \overset{\star}{\rightsquigarrow} A'_{\downarrow \text{pi}} \mathcal{R} \llbracket A'_{\downarrow \text{pi}} \rrbracket_{\downarrow \text{pi}} = \llbracket A' \rrbracket_{\downarrow \text{pi}} = (\llbracket \mathcal{P} \rrbracket_{\downarrow \text{pi}} \cup \{\llbracket P \rrbracket\}_{\downarrow \text{pi}}, \Phi) \overset{\star}{\leftarrow} B'$$

— **case 3.b** : $A = \llbracket B \rrbracket_{\downarrow \text{pi}}$ (and $B = B_{\downarrow \text{pi}}$)

For the same reason as in case 2.b, we can write:

$$B = (Q \cup \{\bar{u}\langle t \rangle.Q\}, \Phi) \quad \text{with } \llbracket Q \rrbracket = P \text{ and } \llbracket Q \rrbracket_{\downarrow \text{pi}} = \mathcal{P}$$

and we conclude by writing $B \xrightarrow{\bar{\xi}\langle a x_n \rangle} B' = (Q \cup \{Q\}, \Phi)$ and:

$$A' \overset{\star}{\rightsquigarrow} (\llbracket Q \rrbracket_{\downarrow \text{pi}} \cup \{\llbracket Q \rrbracket\}_{\downarrow \text{pi}}, \Phi) = \llbracket B' \rrbracket_{\downarrow \text{pi}} = \llbracket B'_{\downarrow \text{pi}} \rrbracket_{\downarrow \text{pi}} \mathcal{R} B'_{\downarrow \text{pi}} \overset{\star}{\leftarrow} B'$$

— **case 4** (rule (COMM)): $\alpha = \overset{\star}{\rightsquigarrow}$ and:

$$\begin{aligned} A &= (\mathcal{P} \cup \{\bar{u}\langle t \rangle.P, v(x).Q\}, \Phi) && \text{with } \text{msg}u, \text{msg}v, \text{msg}t \text{ and } u\downarrow = v\downarrow \\ A' &= (\mathcal{P} \cup \{P, Q\{x \mapsto t\}\}, \Phi) \end{aligned}$$

Then, by a case analysis on the hypothesis $A \mathcal{R} B$:

— **case 4.a** : $B = \llbracket A \rrbracket_{\downarrow \text{pi}}$

Then we can write:

$$B = (\llbracket \mathcal{P} \rrbracket_{\downarrow \rho_i} \cup \{\bar{u}\langle t \rangle. \llbracket P \rrbracket, v(x). \llbracket Q \rrbracket\}, \Phi)$$

and we conclude by remarking that $B \xrightarrow{\tau} B' = (\llbracket \mathcal{P} \rrbracket_{\downarrow \rho_i} \cup \{\llbracket P \rrbracket, \llbracket Q \rrbracket\{x \mapsto t\}\}, \Phi)$ and:

$$A' \overset{\star}{\rightsquigarrow} A'_{\downarrow \rho_i} \mathcal{R} \llbracket A'_{\downarrow \rho_i} \rrbracket_{\downarrow \rho_i} = \llbracket A' \rrbracket_{\downarrow \rho_i} = (\llbracket \mathcal{P} \rrbracket_{\downarrow \rho_i} \cup \{\llbracket P \rrbracket_{\downarrow \rho_i}, \llbracket Q \rrbracket_{\downarrow \rho_i}\{x \mapsto t\}\}, \Phi) \overset{\star}{\leftarrow} B'$$

— **case 4.b** : $A = \llbracket B \rrbracket_{\downarrow \rho_i}$ (and $B = B_{\downarrow \rho_i}$) and a term w such that $w \downarrow = u \downarrow$ appears in B (syntactically)

In particular $\downarrow u$ is not a fresh name introduced by the translation $\llbracket \cdot \rrbracket$ and we can therefore write:

$$B = (\mathcal{P}' \cup \{\bar{u}\langle t \rangle. P', v(x). Q'\}, \Phi) \quad \text{with } \llbracket \mathcal{P}' \rrbracket_{\downarrow \rho_i} = \mathcal{P}, \llbracket P' \rrbracket = P \text{ and } \llbracket Q' \rrbracket = Q$$

and we conclude by writing $B \xrightarrow{\tau} B' = (\mathcal{P}' \cup \{P', Q'\{x \mapsto t\}\}, \Phi)$ and:

$$A' \overset{\star}{\rightsquigarrow} (\llbracket \mathcal{P}' \rrbracket_{\downarrow \rho_i} \cup \{\llbracket P' \rrbracket_{\downarrow \rho_i}, \llbracket Q' \rrbracket_{\downarrow \rho_i}\{x \mapsto t\}\}, \Phi) = \llbracket B' \rrbracket_{\downarrow \rho_i} = \llbracket B'_{\downarrow \rho_i} \rrbracket_{\downarrow \rho_i} \mathcal{R} B'_{\downarrow \rho_i} \overset{\star}{\leftarrow} B'$$

— **case 4.c** : $A = \llbracket B \rrbracket_{\downarrow \rho_i}$ (and $B = B_{\downarrow \rho_i}$) and there exists no term w appearing in B such that $w \downarrow = u \downarrow$ (syntactically)

Then u is a fresh name introduced by $\llbracket \cdot \rrbracket$. We consider the two disjoint cases where it was introduced for the translation of a sum or a circuit:

— If $u \in \mathcal{N}$ is a fresh name generated in order to translate a sum of B , or rephrased more formally:

$$B = (Q \cup \{P' + Q'\}, \Phi)$$

$$A = (\llbracket Q \rrbracket_{\downarrow \rho_i} \cup \{\bar{u}\langle u \rangle, u(x). \llbracket P' \rrbracket, u(x). \llbracket Q' \rrbracket\}, \Phi) \quad \text{where } x \in \mathcal{X}^1 \text{ but } x \notin \text{vars}P', Q'$$

$$A' = (\llbracket Q \rrbracket_{\downarrow \rho_i} \cup \{\llbracket R \rrbracket, u(x). \llbracket S \rrbracket\}, \Phi) \quad \text{where } R, S \in \{P', Q'\}, R \neq S$$

We let $B' = (Q \cup \{R\}, \Phi)$ and remark that $B \xrightarrow{\tau} B'$ by the rule CHOICE (if $R = P'$ and $S = Q'$, or $R = Q'$ and $S = P'$). Besides, let us observe that the name u does not appear in $\llbracket Q \rrbracket_{\downarrow \rho_i}, \llbracket R \rrbracket$ nor Φ by construction of $\llbracket \cdot \rrbracket$ and that $u(x). \llbracket S \rrbracket$ is therefore easily seen to be silent in $A'' = (\llbracket Q \rrbracket_{\downarrow \rho_i} \cup \{\llbracket R \rrbracket\}, \Phi)$. In particular it entails that $A' \rightsquigarrow A''$ by the rule simplification which gives the conclusion:

$$A' \rightsquigarrow A'' \overset{\star}{\rightsquigarrow} (\llbracket Q \rrbracket_{\downarrow \rho_i} \cup \{\llbracket R \rrbracket\}_{\downarrow \rho_i}, \Phi) = \llbracket B' \rrbracket_{\downarrow \rho_i} = \llbracket B'_{\downarrow \rho_i} \rrbracket_{\downarrow \rho_i} \mathcal{R} B'_{\downarrow \rho_i} \overset{\star}{\leftarrow} B'$$

— $u \in \mathcal{N}$ is a fresh name generated in order to translate a Choose(x): this case can be handle analogously to the previous one.

— If $u \in \mathcal{N}$ is a fresh name generated in order to translate a circuit of B , or formally:

$$B = (Q \cup \{\vec{x} \leftarrow \Gamma(\vec{b}).P'\}, \Phi)$$

$$A = (\llbracket Q \rrbracket_{\downarrow_{\text{pi}}} \cup \{\llbracket \vec{x} \leftarrow \Gamma(\vec{b}).P' \rrbracket\}, \Phi)$$

We call $(c_i)_i$ the private fresh names introduced by the translation $\llbracket \vec{x} \leftarrow \Gamma(\vec{b}).P' \rrbracket$, stressing that none of them appears in $\llbracket Q \rrbracket_{\downarrow_{\text{pi}}}$, $\llbracket P' \rrbracket$ nor Φ . Here $u \in \{c_i\}_i$ and we can therefore write:

$$A' = (\llbracket Q \rrbracket_{\downarrow_{\text{pi}}} \cup \{\llbracket Q' \rrbracket\}, \Phi) \quad \text{where } (\{\llbracket \vec{x} \leftarrow \Gamma(\vec{b}).P' \rrbracket\}, \Phi) \xrightarrow{\alpha} (\{\llbracket Q' \rrbracket\}, \Phi)$$

If the sequence \vec{b} contains a term which is not a message or does not reduce to a boolean, then one easily obtain that $(\{\llbracket Q' \rrbracket\}, \Phi) \xrightarrow{\star}_{\text{pi}} (\mathbb{S}, \Phi)$ where \mathbb{S} is silent in $(\llbracket Q \rrbracket_{\downarrow_{\text{pi}}}, \Phi)$ (for that we assume, w.l.o.g. that each input of Γ goes through at least one gate). Hence since $\{\vec{x} \leftarrow \Gamma(\vec{b}).P'\}$ is also silent in (Q, Φ) , we conclude with $B' = (Q, \Phi)$.

Otherwise assume that $\text{msg}\vec{b}$ and $\vec{b} \downarrow \subseteq \mathbb{B}$. Then one easily obtain by induction on the number of gates of Γ that $(\{\llbracket Q' \rrbracket\}, \Phi) \xrightarrow{\star}_{\text{pi}} (\{\llbracket P' \rrbracket\} \{\vec{x} \mapsto \Gamma(\vec{b})\}, \Phi)$ and we conclude by choosing $B' = (Q \cup \{\llbracket P' \rrbracket\} \{\vec{x} \mapsto \Gamma(\vec{b})\}, \Phi)$.

— **case 5** (rules CHOICE, CHOOSE-0, CHOOSE-1 or VALUATE):

The arguments of each of these cases are analogous to priorly-met subcases. ■

In particular note that $A \xrightarrow{\star} A_{\downarrow_{\text{pi}}} \mathcal{R} \llbracket A_{\downarrow_{\text{pi}}} \rrbracket_{\downarrow_{\text{pi}}} = \llbracket A \rrbracket_{\downarrow_{\text{pi}}} \xrightarrow{\star} \llbracket A \rrbracket$ for all extended processes A , hence Lemma D.10.

D.3 Reductions in the pure calculus

We now formalise and prove the correctness of the reductions intuited in Section 5.5.2.

For trace equivalence We define the processes $P(t)$, A and B as follows.

$$P(t) \triangleq c(\vec{x}). \text{Choose}(\vec{y}). v \leftarrow \varphi(\vec{x}, \vec{y}). \bar{c}\langle t \rangle$$

$$A \triangleq P(v) + P(1)$$

$$B \triangleq P(0) + P(1)$$

PROPOSITION D.13 (Reduction for trace equivalence). $A \approx_t B$ iff $\forall \vec{x}. \exists \vec{y}. \varphi(\vec{x}, \vec{y}) = 0$.

PROOF. We do the proof by double implication.

\Rightarrow Suppose that $A \not\approx_t B$. By a quick case analysis, we obtain $B \xrightarrow{\varepsilon} (\{\llbracket P(0) \rrbracket\}, \emptyset) \xrightarrow{\text{tr}} (\emptyset, \{\text{ax}_1 \mapsto 0\})$ where $\text{tr} = c(\vec{t}). \bar{c}\langle \text{ax}_1 \rangle$ for some messages \vec{t} , such that for all reduction $A \xrightarrow{\text{tr}} (C, \Phi)$ the frames $\{\text{ax}_1 \mapsto 0\}$ and Φ are not statically equivalent. In particular, for all $\vec{y} \subseteq \mathbb{B}$, by choosing $\Phi = \{\text{ax}_1 \mapsto \varphi(\vec{t}, \vec{y})\}$ reachable from A , we obtain $\varphi(\vec{t}, \vec{y}) \neq 0$, hence the result.

\Leftarrow Conversely, suppose that exists exists $\vec{x} \subseteq \mathbb{B}$ such that $\varphi(\vec{x}, \vec{y}) = 1$ for all $\vec{y} \subseteq \mathbb{B}$. Then the trace $B \xrightarrow{\varepsilon} (\emptyset, \{ax_1 \mapsto 0\})$ cannot be matched in A and therefore $A \not\approx_t B$. \blacksquare

For simulations We recall that a graphical depiction of the processes has been provided in Section 5.5.2. We fix a family of private channels $(c_P)_P \subseteq \mathcal{N}$ indexed by processes P which will be used to simulate instructions $\text{Goto} \langle P \rangle$. We use a shortcut $\bar{d} \langle \vec{t}^p \rangle$ for an indexed sequence of terms $(t_i)_i$ to denote the sequence of outputs:

$$\bar{d} \langle \vec{t}^p \rangle \triangleq \bar{d} \langle t_1 \rangle \dots \bar{d} \langle t_p \rangle$$

and a similar notation for sequences of inputs. Then the Goto feature is implemented as follows, allowing for passing and receiving program states through parallel processes:

$$\begin{aligned} \text{Goto} \langle A_i \rangle &\triangleq \overline{c_{A_i}} \langle \vec{x}^i, \vec{y}^i \rangle & \text{Goto} \langle B_i \rangle &\triangleq \overline{c_{B_i}} \langle \vec{x}^i, \vec{y}^i \rangle \\ \text{GetEnv} \langle A_i \rangle . P &\triangleq c_{A_i} \langle \vec{x}^i, \vec{y}^i \rangle & \text{GetEnv} \langle B_i \rangle . P &\triangleq c_{B_i} \langle \vec{x}^i, \vec{y}^i \rangle \end{aligned}$$

Formally the processes A_i , B_i and D_i are defined below. We stress out that A and B are closed (as required) but that A_i , B_i and D_i are not in general. Fixing a public channel $c \in \mathcal{F}_0$, we write:

$$\begin{aligned} \forall i \leq n, A_i &\triangleq c(x_i). x_i \leftarrow x_i. D_i \\ \forall i \leq n, B_i &\triangleq c(x_i). x_i \leftarrow x_i. (D_i + (c(y_i). y_i \leftarrow y_i. \text{Goto} \langle B_{i+1} \rangle)) \\ A_{n+1} &\triangleq v \leftarrow \varphi(\vec{x}, \vec{y}). \bar{c} \langle v \rangle \\ B_{n+1} &\triangleq \bar{c} \langle 0 \rangle \\ D_i &\triangleq \text{Choose}(z_i). c(y_i). r_i \leftarrow (y_i = z_i). \\ &((\text{if } r_i = 1 \text{ then } \text{Goto} \langle A_{i+1} \rangle) \\ &| (\text{if } r_i = 0 \text{ then } c(y_i). y_i \leftarrow y_i. \text{Goto} \langle B_{i+1} \rangle)) \end{aligned}$$

As in the reduction for trace equivalence, the $\text{Choose}(\alpha)$ simulates non-deterministic choice among \mathbb{B} ; the construction $\alpha \leftarrow \alpha$, which may seem useless, encodes the test $\alpha \in \mathbb{B}$. Finally, we define A and B by putting the auxiliary processes in parallel and connecting the Goto's to the getEnv's:

$$A \triangleq A_1 \mid C \quad B \triangleq B_1 \mid C \quad C \triangleq \prod_{i=2}^{n+1} (\text{GetEnv} \langle A_i \rangle . A_i) \mid \prod_{i=2}^{n+1} (\text{GetEnv} \langle B_i \rangle . B_i)$$

A and B can be computed in time $O(n^2 + |\varphi|)$ in a straightforward way. The formal proof of the reduction is detailed below.

PROPOSITION D.14 (Correctness of the reduction). *The following statements are equivalent:*

1. $A \approx_b B$
2. $A \approx_s B$
3. $\forall x_1 \exists y_1 \dots \forall x_n \exists y_n. \varphi(x_1, \dots, x_n, y_1, \dots, y_n) = 0$

PROOF. We recall that, again, our proof uses the advanced winning strategy framework presented in Section D.1.

1 \Rightarrow 2 Follows from the inclusion $\approx_b \subset \approx_s$.

3 \Rightarrow 1 Suppose that $\forall x_1 \exists y_1 \dots \forall x_n \exists y_n. \varphi(x_1, \dots, x_n, y_1, \dots, y_n) = 1$. For convenience we use a notation for subprocess extraction: if ℓ is a position of a process C , then the subprocess of C at position ℓ (which may not be closed) is denoted by $C|_\ell$. Then writing in addition:

$$C_i = \prod_{j=i+1}^{n+1} (\text{GetEnv} \langle A_j \rangle . A_j) \mid \prod_{j=i+1}^{n+1} (\text{GetEnv} \langle B_j \rangle . B_j)$$

we define \mathcal{R} the smallest reflexive symmetric relation on closed extended processes such that:

1. $(A_i \mid C_i)(\vec{x}^{i-1}, \vec{y}^{i-1}) \mathcal{R} (B_i \mid C_i)(\vec{x}^{i-1}, \vec{y}^{i-1})$
if $\forall x_i \exists y_i \dots \forall x_n \exists y_n. \varphi(\vec{x}, \vec{y})$.
2. $(A_{i|\ell} \mid C_i)(\vec{x}^i, \vec{y}^{i-1}) \mathcal{R} (B_{i|\ell} \mid C_i)(\vec{x}^{i-1}, \vec{y}^{i-1})$
if $\ell \in \{0, 0.0\}$ and $\exists y_i \dots \forall x_n \exists y_n. \varphi(\vec{x}, \vec{y})$.
3. $(B_{i|0.0.1.\ell} \mid C_i)(\vec{x}^i, \vec{y}^i) \mathcal{R} (D_{i|0.\ell} \mid C_i)(\vec{x}^i, \vec{y}^i)$
if $\ell \in \{\varepsilon, 0\}$ and $\forall x_{i+1} \exists y_{i+1} \dots \forall x_n \exists y_n. \varphi(\vec{x}, \vec{y})$.

Then one can verify that \mathcal{R} is a bisimulation up to \rightsquigarrow , and $A \mathcal{R} B$ by hypothesis, hence the $A \approx_b B$.

2 \Rightarrow 3 By contraposition, if we suppose that $\exists x_1 \forall y_1 \dots \exists x_n \forall y_n. \varphi(x_1, \dots, x_n, y_1, \dots, y_n) = 1$, then one can define a labelled attack \mathcal{S} (valid against simulation) such that $B \mathcal{S} A$. We omit the concrete construction as it is analogous to that of \mathcal{R} above; all in all this gives the conclusion $A \not\approx_s B$. ■

D.4 Reduction in the full calculus

Before concretely proving the pending lemmas, let us introduce some notations and prove intermediary results about static equivalence. We fix a private nonce $s \in \mathcal{N}$ and define the following frames given a protocol term t :

$$\begin{aligned} \Phi_t &= \{ax_1 \mapsto h(t, s), ax_2 \mapsto h(1, s)\} \\ \Phi_t^{\mathcal{N}} &= \{ax_1 \mapsto h_{\mathcal{N}}(t, s), ax_2 \mapsto h(1, s)\} \\ \Phi_t^{\mathbb{B}} &= \{ax_1 \mapsto h_{\mathbb{B}}(t, s), ax_2 \mapsto h(1, s)\} \end{aligned}$$

One shall observe that the whole deal with our reduction is about which instances of these three frames are reachable in which conditions. Hence first we prove a lemma investigating the static equivalence between some of them:

LEMMA D.15. *Let t be a message in normal form ($t = t\downarrow$). The following properties hold:*

- (i) $\Phi_t \sim \Phi_0$ iff $t \neq 1$
- (ii) $\Phi_t^N \sim \Phi_0$ iff $\text{root}(t) \neq \text{Node}$
- (iii) $\Phi_t^B \sim \Phi_0$ iff $t \notin \mathbb{B}$

PROOF. We prove the three equivalences together by double implication.

(\Rightarrow) We prove the three properties by contraposition. We naturally proceed by exhibiting ground recipes ξ, ζ witnessing the non-static-equivalence goal:

- (i) We assume $t = 1$ and we choose $\xi = ax_1$ and $\zeta = ax_2$: the conclusion follows from $\xi\Phi_t\downarrow = h(1, s) = \zeta\Phi_t\downarrow$ and $\xi\Phi_0\downarrow = h(0, s) \neq h(1, s) = \zeta\Phi_0\downarrow$.
- (ii) We assume $t = \text{Node}(t_1, t_2)$ and we choose $\xi = \text{TestNode}(ax_1)$: the conclusion follows from $\text{msg}\xi\Phi_t^N$ and $\neg\text{msg}\xi\Phi_0^N$.
- (iii) We assume $t \in \{0, 1\}$ and we choose $\xi = \text{TestBool}(ax_1)$: the conclusion follows from $\text{msg}\xi\Phi_t^B$ and $\neg\text{msg}\xi\Phi_0^B$.

(\Leftarrow) The key point is an observation about rewriting critical pairs (not specific to \mathcal{R}):

if: u is a term;

if: σ is a substitution such that for any $m \in \text{img}(\sigma)$ there exists no rule $\ell \rightarrow r$ of \mathcal{R} such that m is unifiable with a subterm $u \in \text{st}\ell - \mathcal{X}$;

then: for any rewriting sequence $u\sigma \rightarrow^* s$, it holds that $s = u'\sigma$ for some $u \rightarrow^* u'$ (where u' is in normal form iff s is in normal form. In particular $(u\sigma)\downarrow = (u\downarrow)\sigma$).

One shall note that any frame Φ investigated by the lemma (Φ_t when $t \neq 1$, Φ_t^N when $\text{root}t \neq \text{Node}$ and Φ_t^B when $t \notin \mathbb{B}$) verifies the second hypothesis. As a consequence, if ξ is a ground recipe such that $\text{axioms}\xi \subseteq \text{dom}(\Phi)$, then $\text{msg}\xi\Phi$ iff $\text{msg}\xi\Phi_0$ (iff $\forall \zeta \in \text{st}\xi, \zeta\downarrow \in \mathcal{T}(\mathcal{F}_c, \mathcal{F}_0 \cup \mathcal{AX})$). This settles the first item of the definition of static equivalence. As for the second item, let us fix two ground recipes ξ, ζ such that $\text{msg}\xi\Phi$ and $\text{msg}\xi\Phi_0$. Let us then prove that $\xi\Phi\downarrow = \zeta\Phi\downarrow$ iff $\xi\Phi_0\downarrow = \zeta\Phi_0\downarrow$ by induction on $(\xi\downarrow, \zeta\downarrow)$. Note that we will intensively (and implicitly) use the fact that $\xi\Phi\downarrow = (\xi\downarrow)\Phi$ (same for ζ and/or Φ_0).

- **case 1** : $\xi\downarrow = f(\xi_1, \dots, \xi_n)$ and $\zeta\downarrow = f(\zeta_1, \dots, \zeta_p)$ with $f, g \in \mathcal{F}_c$.

If $f = g$ then the result follows from induction hypothesis and if $f \neq g$ the conclusion is immediate ($\xi\Phi\downarrow \neq \zeta\Phi\downarrow$ and $\xi\Phi_0\downarrow \neq \zeta\Phi_0\downarrow$).

- **case 2** : $\xi\downarrow \in \mathcal{AX}$ and $\zeta\downarrow \in \mathcal{AX}$.

If $\xi = \zeta$ the conclusion is immediate and so is it when $\xi \neq \zeta$ since $\Phi(ax_1) \neq \Phi(ax_2)$ and $\Phi_0(ax_1) \neq \Phi_0(ax_2)$.

- **case 3** : $\xi\downarrow \in \mathcal{AX}$ and $\zeta\downarrow = f(\zeta_1, \dots, \zeta_p)$ with $f \in \mathcal{F}_c$.

We argue that $\xi\Phi\downarrow \neq \zeta\Phi\downarrow$ and $\xi\Phi_0\downarrow \neq \zeta\Phi_0\downarrow$. Either of the two equalities being verified would indeed imply that $s \in \{\zeta_2\Phi, \zeta_2\Phi_0\}$: this is impossible as ζ_2 is a ground recipe in normal form and $s \in \mathcal{N}$ (one easily shows that $\zeta_2\Phi$ and $\zeta_2\Phi_0$ are either public names, constants, or terms of height 1 or more).

As $\text{msg}\xi\Phi$ and $\text{msg}\xi\Phi_0$, the preliminary observation justifies that $\xi\downarrow$ and $\zeta\downarrow$ function symbols are all constructor. In particular no other cases than the three above need to be considered, which concludes the proof. ■

With this lemma in mind, the proofs of Propositions 5.21 and 5.22 become quite straightforward:

PROPOSITION 5.21 (correctness of the tree checker). *Let x be a message which is not a complete binary tree of height n with boolean leaves. Then there exists a reduction $\text{CheckTree}(x) \xrightarrow{\varepsilon} (\llbracket P \rrbracket, \emptyset)$ such that $P \approx_b \bar{c}\langle h(0, s) \rangle. \bar{c}\langle h(1, s) \rangle$.*

PROOF. Let x be a message which is not a complete binary tree of height n whose leaves are booleans.

case 1 : there exists a position $\vec{\pi} \in \mathbb{B}^*$ such that $|\vec{\pi}| = i \in \llbracket 0, n-1 \rrbracket$ and $\text{root}x_{|\vec{\pi}} \neq \text{Node}$.

The result follows from Lemma D.15 after writing the following sequence of transitions:

$$\begin{aligned} \text{CheckTree}(x) &\xrightarrow{\varepsilon} \text{Choose}(p_1, \dots, p_i). \bar{c}\langle h_{\mathbb{N}}(x_{|p_1 \dots p_i}, s) \rangle. \bar{c}\langle h(1, s) \rangle \\ &\xrightarrow{\varepsilon} \bar{c}\langle h_{\mathbb{N}}(x_{|\vec{\pi}}, s) \rangle. \bar{c}\langle h(1, s) \rangle \end{aligned}$$

case 2 : there exists a position $\vec{\pi} \in \mathbb{B}^n$ such that $x_{|\vec{\pi}} \notin \mathbb{B}$.

The result follows from Lemma D.15 after writing the following sequence of transitions:

$$\begin{aligned} \text{CheckTree}(x) &\xrightarrow{\varepsilon} \text{Choose}(p_1, \dots, p_n). \bar{c}\langle h_{\mathbb{N}}(x_{|p_1 \dots p_n}, s) \rangle. \bar{c}\langle h(1, s) \rangle \\ &\xrightarrow{\varepsilon} \bar{c}\langle h_{\mathbb{B}}(x_{|\vec{\pi}}, s) \rangle. \bar{c}\langle h(1, s) \rangle \end{aligned} \quad \blacksquare$$

PROPOSITION 5.22 (correctness of the sat checker). *Let x be a complete binary tree of height n whose leaves are booleans, and val_x be the valuation mapping the variable number i of $\llbracket \Gamma \rrbracket_\varphi$ to $x_{|p_1 \dots p_n} \in \mathbb{B}$ where $p_1 \dots p_n$ is the binary representation of i (i.e., $i = \sum_{k=1}^n p_k 2^{k-1}$). If val_x does not satisfy $\llbracket \Gamma \rrbracket_\varphi$ then there exists $\text{CheckSat}(x) \xrightarrow{\varepsilon} P$ such that $P \approx_b \bar{c}\langle h(0, s) \rangle. \bar{c}\langle h(1, s) \rangle$.*

PROOF. Let x be a complete binary tree of height n whose leaves are booleans, that is to say, a message such that $x_{|\vec{p}} \in \mathbb{B}$ for all $\vec{p} \in \mathbb{B}^n$. Naming x_0, \dots, x_{2^n-1} the variables of $\llbracket \Gamma \rrbracket_\varphi$ in this order, val_x refers to the valuation mapping x_i to $x_{|\vec{p}}$ where \vec{p} is the binary representation of i (of size n with padding head 0's).

Let us now assume that val_x does not satisfy $\llbracket \Gamma \rrbracket_\varphi$. In particular there exists a clause of $\llbracket \Gamma \rrbracket_\varphi$, say the i^{th} clause with $i = \sum_{k=1}^m \pi_k 2^{k-1}$, which is falsified by val_x . In particular, if the three variables of this clause are called x_1, x_2, x_3 with respective negation bits b_1, b_2, b_3 , the following formula is evaluated to false (i.e. 0):

$$\bigvee_{i=1}^3 (b_i = \text{val}_x(x_i))$$

Therefore, by choosing the sequence π_1, \dots, π_m to instantiate the initial $\text{Choose}(p_1, \dots, p_m)$ of $\text{CheckSat}(x)$, we obtain the following sequence of transitions, which concludes the proof:

$$\text{CheckSat}(x) \xrightarrow{\varepsilon} \bar{c}\langle h(0, s) \rangle. \bar{c}\langle h(1, s) \rangle \quad \blacksquare$$

We finally gathered all the ingredients needed to prove the main lemma:

PROPOSITION 5.23 (correctness of the reduction). *For any equivalence relation $\approx \in \{\approx_s, \approx_t, \approx_b\}$, $\llbracket \Gamma \rrbracket_\varphi$ is satisfiable iff $A \approx B$.*

PROOF. We prove the result by double implication.

\Rightarrow Let us consider a valuation satisfying φ and let t be a message such that val_t is equal to this valuation. Then since the trace $B \xrightarrow{c(t).\bar{c}\langle ax_1 \rangle.\bar{c}\langle ax_2 \rangle} (\emptyset, \Phi_0)$ cannot be matched in A by Lemma D.15, we obtain $B \not\sqsubseteq_t A$. Hence, since trace equivalence is the coarsest of the considered equivalence relations, we obtain the desired result.

\Leftarrow By contraposition, let us suppose that φ is unsatisfiable, and let us prove that $A \approx_b B$ (which implies all other equivalences). Let us consider \mathcal{R} the smallest reflexive symmetric relation on extended processes such that:

1. $A \mathcal{R} B$
2. $A'(x) \mathcal{R} B'(x)$ for all message x , where $A = c(x).A'(x)$ and $B = c(x).B'(x)$
3. $P_i \mathcal{R} P'_i$, where:

$$P_i = (\{\bar{c}\langle t_{i+1} \rangle \dots \bar{c}\langle t_p \rangle\}, \{ax_1 \mapsto t_1, \dots, ax_i \mapsto t_i\})$$

$$P'_i = (\{\bar{c}\langle t'_{i+1} \rangle \dots \bar{c}\langle t'_p \rangle\}, \{ax_1 \mapsto t'_1, \dots, ax_i \mapsto t'_i\})$$

and where $\{ax_1 \mapsto t_1, \dots, ax_p \mapsto t_p\} \sim \{ax_1 \mapsto t'_1, \dots, ax_p \mapsto t'_p\}$

It easily follows from Lemma D.15, 5.21, 5.22 that \mathcal{R} is a bisimulation up to \rightsquigarrow , hence the conclusion. ■