Automated VAlidatioN of Trust and Security of Service-oriented ARchitectures

FP7-ICT-2007-1, Project No. 216471

www.avantssar.eu

Deliverable D3.4
Abstraction and Compositional Reasoning Techniques for Service Analysis

Abstract

We present a number of abstraction techniques for the validation of trust and security properties of services. Abstraction consists in transforming a *concrete* model to be analyzed into a *abstract* model more amenable to analysis. The transformation must be sound w.r.t. a class of properties, i.e. a property proved on the abstract model must also hold in the concrete one. This implies that the original model must be over-approximated, i.e. every reachable state or trace of the concrete system has a counterpart (modulo an abstraction relation) in the abstract system. Thus, in the worst case, a concrete system may be secure, while we are not able to prove this using its abstraction.

Deliverable details

Deliverable version: *v1.0* Classification: *public*
Date of delivery: *31.10.2010* Due on: *31.10.2010*
Editors: *UNIVR, ETH Zurich, INRIA, UPS-IRIT, IBM (principal editors). UGDIST, OpenTrust, IEAT, SAP, SIEMENS (secondary editors)* Total pages: *94*

Project details

Start date: *January 01, 2008* Duration: *36 months*
Project Coordinator: *Luca Viganò*
Partners: UNIVR, ETH Zurich, INRIA, UPS-IRIT, UGDIST, IBM, OpenTrust, IEAT, SAP, SIEMENS

Contents

List of Figures

List of Tables

1 Introduction

Abstraction consists in transforming the model to be analysed (a *concrete* model) into a simpler one (an *abstract* model) amenable to analysis. This transformation can be defined on the states of the model (data abstraction) or on the relation defining the possible transitions between states of the model (control abstraction), or both. One requires that the transformation must be sound with respect to a class of properties, i.e. a property proved on the transformed model must also hold in the original one. This requirement implies that the original model must be over-approximated in the sense that every reachable state or trace of the concrete system has a counterpart (modulo an abstraction relation) in the abstract system. Thus, in the worst case, a concrete system may be secure, while we are not able to prove this using its abstraction. In the analysis of security protocols, data abstraction has been widely used to verify protocols for an unbounded number of sessions, e.g. [17, 22, 26, 33, 42, 52, 61], mapping the infinite set of fresh data created during the sessions to a finite set, and giving sufficient conditions for this abstraction to be correct [27]. The transitions can also be abstracted by a set of Horn clauses. Hence, the number of times a transition can be fired as well as the ordering of transitions is relaxed. This abstraction amounts to considering an over-approximation of the reachability relation that does not suffer from the classical interleaving problems encountered in concurrent systems verification. Both abstraction techniques can be fruitfully employed for scaling services validation procedures too. In this deliverable, we present several abstraction techniques for the validation of trust and security properties of services.

Set-Based Abstraction Previously known Horn-clause protocol abstractions fail on problems where the set of true facts does not monotonically grow with the executions of the protocols. However, such situations frequently occur when servers maintain some form of database, for instance a key-server maintains a set of keys, to which agents they belong and what their status is, e.g. valid or revoked/outdated. We introduce a new abstraction technique (§ 2) for tackling this problem. We define an equivalence relation on all created data according to their status and membership in the databases of the participants. For instance, suppose there are two agents a and b which each maintain a set of keys that are either unknown, valid, or revoked; two concrete keys k_1 and k_2 are now mapped to the same abstract key k iff they are equal in the membership of the databases, e.g. a considers both keys as revoked, and b considers both as valid. This approach is practically feasible. The implementation and a library of examples with detailed descriptions is

available in [54].

Encoding security-policy clauses To decide reachability-based properties in security-sensitive services we need to also take the policy engines of services into account. Towards a seamless decision algorithm that accounts for both the communication and policy levels of services, we propose in § 3 an encoding that shows that (a fragment of) policy-level computations of services can be seen as message derivations in the Dolev-Yao attacker model. Then, intuitively, the attacker and all the services would be equipped with the reasoning power of the Dolev-Yao model, which is well understood and comes with decision algorithms for reachability. The approach has been applied to policies expressing trust application or trust delegation, as well as role hierarchy as in Role based access control.

One-step Transition Decision Procedures In ASLan, a transition is defined modulo a set of Horn clauses that express the security policies and trust-negotiation capabilities of the individual entities. Our choice of Horn clauses enables one to encode rich security policies, but this chosen setting implies that deciding whether the security policies of the different entities allow for the execution of a given step in ASLan is in general undecidable. We present in § 4 some restrictions or assumptions under which one regains decidability while keeping an acceptable level of expressiveness.

Compositional reasoning is important too in software engineering, e.g. for modular software design and for stepwise refinement. For instance, one may validate a system that comprises partially specified components, therefore avoiding too early commitments. Compositional reasoning has already been addressed in Deliverable 3.3 [7], where several abstractions of communication channels have been proposed and applied to SAML-SSO. We pursue in this deliverable the development of suitable compositional reasoning techniques for services.

Process Equivalence Formal methods and related tools have proved to be successful for checking that protocols and services satisfy some security properties. But they are limited in expressiveness since in most cases the focus has been on the resolution of reachability problems, and as a consequence very few effective procedures (e.g. [18]) consider the more general case of equivalence properties useful for modelling many important security properties. For instance, the fundamental security compositionality results by Canetti et al. (e.g., [28, 48]) require one to prove that a process is equivalent to an ideal abstraction of it in every environment. Hence, deciding

equivalence of processes is mandatory to obtain automated compositionality proofs. In § 5, we give an algorithm for deciding the equivalence of processes represented as symbolic constraints. The result applies to the large class of deduction systems parameterized by a subterm convergent equational theory.

2 Set-Based Abstraction

Tools based on over-approximation like ProVerif have been very successful on the verification of security protocols and web services [17, 25, 23, 61, 16]. In contrast to conventional model checking approaches like [50, 11, 5], the over-approximation methods do not consider a state transition system, but just a set of derivable (state-independent) facts like intruder knowledge (and the intruder never forgets). Moreover, the creation of fresh keys and nonces is replaced by a function of the context in which they are created. For instance if agent a creates a nonce for use with agent b, this may simply be $n(a, b)$ in every run of the protocol. The main advantage is that this kind of verification works for an unbounded number of sessions, while standard model checking methods consider a bounded number of sessions. In fact, the entire interleaving problem of model checking does not occur in the over-approximation approach, and tools thus also scale better with the number of protocol steps and repeated parts of the protocol. Another advantage is that models of this kind can be represented as a set of first-order Horn clauses for which many existing methods can be used off the shelf, e.g. the SPASS theorem prover [61, 62].

A disadvantage of the abstractions are false positives, i.e. attacks that are introduced by the over-approximation. In the worst case we may thus fail to verify a correct protocol. This problem can sometimes be solved by refining the abstraction. However, if we turn to more complex systems that consist of several protocols or web services, the abstraction approaches reach a limitation. The reason is that we may consider servers that maintain some form of database, for instance a key-server maintains a set of keys, to which agents they belong and what their status is, e.g. valid or revoked/outdated. Another example is a web service for online shopping that maintains a database of orders that have been processed and their current status. Further, servers may maintain a database of access rights and access rights may be revoked. Common between these examples is that the set of true facts does not monotonically grow with the executions of the protocols. Such non-monotonic behavior simply cannot be expressed in the standard (stateless) approach of abstracting protocols by a set of Horn clauses, because deduction is monotonic, i.e. adding facts like a revocation can never lead to fewer deductions.

This work tackles this problem with a different kind of abstraction of the fresh data while maintaining the basic approach of over-approximating the protocol or web service by a set of first-order Horn clauses. As a basis, we consider a model where each participant can maintain a database in which freshly generated data like nonces, keys, or order-numbers can be stored along with their context, e.g. the owner and status of a key. To deal with such

systems in an abstraction approach, we define the abstraction of all created data by their status and membership in the databases of the participants. For instance, suppose there are two agents a and b which each maintain a set of keys that are either unknown, valid, or revoked; two concrete keys k_1 and k_2 are now mapped to the same abstract key k iff they are equal in the membership of the databases, e.g. a considers both keys as revoked, and b considers both as valid.

So, as usual in these approaches, the infinite set of data is mapped to finitely many equivalence classes or representatives (if we have finitely many participants), but here the abstraction depends on the current state of the databases. Consider for example that the intruder knows a message m containing, as a subterm, the abstract key k mentioned above. Consider further a transition rule that allows one to revoke a key at agent b, so that in the abstract model, the key k should be "transformed" into a key k' representing all the keys that are known as revoked to both a and b. The idea to handle this in the abstraction is to maintain all previous facts that contain the key k in its old form and to add also all these facts with k' replaced for k. So everything the intruder knows with a valid key k (in b's eyes), he also knows with a revoked key k'. The intuitive reason why this is indeed sound is that—thanks to the over-approximation—every derivation in the abstract model corresponds to an unlimited number of executions with concrete data that fall into the same equivalence class.

The transformation of facts that arises from the state-transition of the database is expressed by a new kind of rule, so-called *term implication rules* that have the form $\phi \rightarrow k \twoheadrightarrow k'$. This expresses that, if the clauses in ϕ hold, then $f[k]$ implies $f[k']$ for every context $f[\cdot]$. We show that these rules can be encoded into standard Horn clauses.

The contributions of the work described in this section are both theoretical and practical. First, we define the specification language AIF, a variant of the projects specification language ASLan, that allows for a declarative specification of the un-abstracted transition system with fresh data and databases. Defining AIF has the advantage that we have a clear boundary of the specifications that our method can support, both including some restrictions and some extensions. Second, we define a novel way to abstract this specification into a set of Horn clauses and term implication rules, a concept that naturally arises from this kind of specification. Third, we show that this abstraction is sound, i.e. without excluding attacks. Fourth we show how to encode also the term implication rules as Horn clauses without excluding or introducing attacks. Fifth, we implement this translation from AIF to Horn clauses for the syntax of the tools SPASS and ProVerif, both of which implement state of the art resolution techniques for first-order (Horn) clauses. This allows us

to demonstrate with a number of non-trivial examples that the approach is practically feasible. The implementation and a library of AIF examples with detailed descriptions is available [54].

2.1 AIF and the Concrete Model

We now introduce the language AIF that we use for specifying security protocols, web services, and their goals without the abstraction. It is a variant of ASLan influenced by the needs of our methods and adding syntactic sugar for convenience.

2.1.1 A Running Example

Before we give the formal definition, we first introduce a simple example that we use throughout this section. For simplicity, we limit the example to three agents: the honest user a, the honest server s, and the dishonest intruder i. The full specification considered in subsection 2.6 is parametrized and can be used with any (but fixed) number of honest and dishonest users (see also [54]).

Each agent has a database of its own that contains all the information that this agent has to maintain over a longer time (i.e., that may span several sessions). In our example, the user keeps a database of all its valid public/private key pairs that it currently has registered with the server s. We denote with $\mathsf{inv}(k)$ the private key that belongs to public key k. Thus, all entries of a's database are of the form $(k, \mathsf{inv}(k))$ and it is sufficient to represent the database entries only by the public key k (omitting $\mathsf{inv}(k)$ in the term representation of the database). We thus write the set condition $k \in \mathit{ring}(a)$ for every key k in the database of a.

The server stores in its database the registered keys along with their owner and status, which is either *valid* or *revoked*. One could write for instance $(k, a, \mathit{valid}) \in \mathit{db}(s)$ for a key k that is stored in the database of s as a valid key owned by a, but we rather use a slightly different representation and write $k \in \mathit{db}(s, a, \mathit{valid})$. This is equivalent to thinking of a server that maintains for each user two databases, namely the sets of valid and revoked keys. This representation is helpful for the abstraction below because all sets contain only data that can be abstracted (public keys in this example) rather than a mixture of different kinds of data.

An AIF specification describes a state transition system by a set of rules. The first rule of our example is an initialization rule that represents an out of band registration of the key with a server. (Suppose the user physically

visits the organization that owns the server.)

$$=[PK]\Rightarrow PK \in ring(a) \cdot PK \in db(s, a, valid) \cdot \mathsf{iknows}(PK)$$

This rule can be taken in any state (because there are no conditions left of the arrow) and will first create a fresh value (that never occurred before) that we bind to the variable PK, intuitively a public key. In the successor state, PK is both in the databases of a and of the server as a valid key. We use $\mathsf{iknows}(m)$ to denote that the intruder knows m, so in this case he learns immediately the new public key PK. The rule can be applied any number of times to register as many keys as desired. Note that $\mathsf{iknows}(\cdot)$ does *not* have a predefined meaning in AIF, is rather characterized by intruder deduction rules reflecting the standard Dolev-Yao model, e.g. $\mathsf{iknows}(M).\mathsf{iknows}(\mathsf{inv}(K)) \Rightarrow \mathsf{sign}_{\mathsf{inv}(K)}(M)$ (which can be applied to any state that contains facts matching what is left of $\Rightarrow$).

The second rule of our example is the transmission of a new key using a registered valid key:

$$\begin{array}{l} PK \in ring(a) \cdot \mathsf{iknows}(PK) \\ =[NPK]\Rightarrow NPK \in ring(a) \cdot \mathsf{iknows}(\mathsf{sign}_{\mathsf{inv}(PK)}(new, a, NPK)) \end{array}$$

We do not repeat the condition $PK \in ring(a)$ on the RHS; in AIF this means that this condition gets removed by the transition, i.e. the user a forgets the key PK (which is a bit unrealistic and only done for the sake of simplicity).

The third rule is the server receiving such a message, registering the new key and revoking the old key:

$$\begin{array}{l} \mathsf{iknows}(\mathsf{sign}_{\mathsf{inv}(PK)}(new, a, NPK)) \cdot PK \in db(s, a, valid) \cdot \\ NPK \notin db(s, a, valid) \cdot NPK \notin db(s, a, revoked) \\ \Rightarrow PK \in db(s, a, revoked) \cdot NPK \in db(s, a, valid) \\ \cdot \mathsf{iknows}(\mathsf{inv}(PK)) \end{array}$$

Here, the intruder learns the private key of the revoked key.

To define a security goal, we give yet a further rule that produces the fact *attack* if the intruder finds out the private key of a valid public key of a:

$$\mathsf{iknows}(\mathsf{inv}(PK)) \cdot PK \in db(s, a, valid) \Rightarrow attack$$

2.1.2 Formal Definition of AIF

We use a standard term model of messages, the only specialty is the distinction of constants and variables that will be abstracted later.

Definition 1 Messages *are represented as terms over a signature* $\Sigma \cup \mathfrak{A}$ *and a set* $\mathcal{V}$ *of variables, where* Σ *is finite,* $\mathcal{V}$ *is countable, and* $\mathfrak{A}$ *is a countable set of constant symbols (namely those that are going to be abstracted later).* Σ, $\mathfrak{A}$, *and* $\mathcal{V}$ *are non-empty and pairwise disjoint. Let* $\mathcal{V}_{\mathfrak{A}} \subset \mathcal{V}$ *be a set of variables that can only be substituted by constants of* $\mathfrak{A}$. *Let* $\mathcal{T}_{\mathfrak{A}} = \mathfrak{A} \cup \mathcal{V}_{\mathfrak{A}}$ *denote the set of all* abstractable *symbols. By convention, we use upper-case letters for variables and lower-case letters for constant and function symbols.*

Note that we assume a free algebra interpretation of terms (i.e. two terms are equal iff they are syntactically equal). We come back later to this issue when we use SPASS (which does not consider a fixed interpretation).

Definition 2 *Let* Σ_f *be a finite signature (disjoint from all sets above) of* fact symbols. *A* fact *is a term of the form* $f(t_1, \ldots, t_n)$ *where* f *is a fact symbol of arity* n *and the* t_i *are messages. A* positive (negative) set condition *has the form* $t \in M$ ($t \notin M$) *where* $t \in \mathcal{T}_{\mathfrak{A}}$ *and* M *is a* set expression, *namely a ground message term in which no symbol of* $\mathcal{T}_{\mathfrak{A}}$ *occurs.*

The syntactic form of set expressions like M in this definition enforces that a specification can only use a fixed number of sets that we denote with N. Also, we will thus simply assume these sets are called $s_1, \ldots, s_N$, while in AIF specifications, one we will use more intuitive terms like $ring(a)$ for the set of keys known by agent a.

We now come to the core of the AIF specifications, namely the state transition rules.

Definition 3 *A* state *is a finite set of facts and positive set conditions. A* transition rule r *has the form*

$$LF \cdot S_+ \cdot S_- \; =[F]\Rightarrow \; RF \cdot RS$$

where LF *and* RF *are sets of facts,* S_+ *and* RS *are sets of positive set conditions,* S_- *is a set of negative set conditions, and* $F \subseteq \mathcal{V}_{\mathfrak{A}}$. *We require that*

$$vars(RF \cdot RS \cdot S_-) \subseteq F \cup vars(LF \cdot S_+) \text{ and } vars(S_-) \cap F = \emptyset \; .$$

Moreover, we require that each $t \in \mathcal{T}_{\mathfrak{A}}$ *that occurs in* S_+ *or* S_- *also occurs in* LF *and each* $t \in \mathcal{T}_{\mathfrak{A}}$ *that occurs in* RS *also occurs in* RF.[1]

We say $S \Rightarrow_r S'$ *iff there is a grounding substitution* σ *(for all variables of* r*) such that*

[1] This condition ensures that when we remove set conditions in rules and states in the abstract model below, the elements (that will carry the set conditions in their abstraction) still appear in the normal facts.

- $(LF \cdot S_+)\sigma \subseteq S$,
- $S_-\sigma \cap S = \emptyset$, $S' = (S \setminus S_+\sigma) \cup RF\sigma \cup RS\sigma$,
- $F\sigma$ *are fresh constants from* $\mathfrak{A}$ *(i.e. they do not occur in* S *or any rule* r *that we consider).*

A state S *is called* reachable *using the set of transition rules* R*, iff* $\emptyset \Rightarrow_R^* S$*. Here* $\Rightarrow_R$ *is the union of* $\Rightarrow_r$ *for all* $r \in R$ *and* $\cdot^*$ *is the reflexive transitive closure. (We generally use the* $\emptyset$ *as the initial state.)*

Intuitively, the left-hand side of a rule describes to which states the rule can be applied, and the right-hand side describes the changes to the state after the transition.

There is a subtle difference to ASLan (and to other set-rewriting/multi-set rewriting approaches). In AIF, facts are *persistent*, i.e. a fact that holds in one state also holds in all successor states. The only entities that can be removed from a state during a transitions are the positive set conditions, namely by a transition rule that has a positive condition $x \in s_i$ on the left-hand side that is not repeated on the right-hand side.

The persistence of facts is a restriction with respect to other approaches, but one that comes without loss of generality: a non-persistent fact $f(t_1, \ldots, t_n)$ of ASLan can be simulated in AIF by a persistent fact $f'(t_1, \ldots, t_n, FID)$, where FID is a fresh identifier created when introducing the fact, and using a distinguished set *valid* that contains FID in exactly those states where $f(t_1, \ldots, t_n)$ holds.

Our construction to make set membership the only "revocable" entity while facts monotonously grow over transitions gives a distinction that becomes valuable in the abstraction later. To see that, consider that the AIF transition rules (or the ASLan transition rules) are not monotonic (i.e. a rule that is applicable to a state S is not necessarily applicable to any superset of S). In contrast, the Horn-clauses of the abstract model are interpreted in standard–monotone–first-order logic. Our construction thus ensures that all the non-monotonic aspects, the set memberships, are part of the abstraction.

We close this discussion with the remark that all previous abstraction approaches in protocol verification like [17, 25, 23, 61] are entirely based on persistent facts. This (usually) means an over-approximation that leads to the following phenomenon [53]: every participant can react to a given message any number of times, even if the real system prevents that with challenge-response or timestamps. As can be seen by the success of the abstraction methods, this over-approximation usually works fine (if one does not consider replay which requires special care [19]). So in general, for what

concerns this new abstraction approach where we have the choice to make things revocable, one may start with a model where all facts are persistent and perform the above encoding of non-persistent facts only when necessary, i.e. when one obtains false attacks caused by the over-approximation.

2.1.3 Syntactic Sugar

For readability and brevity of specifications, the AIF language supports a number of constructs to avoid finite enumerations. One can declare a number of variables that range over a given set of constants, e.g.:

$$\begin{array}{rcl} A, B & : & \{a, b, s, i\}; \\ \mathit{Honest} & : & \{a, b\}; \\ \mathit{Status} & : & \{\mathit{valid}, \mathit{revoked}\}; \end{array}$$

We call variables that have been declared in this way *enumeration variables*. An AIF specification includes the enumeration of all sets or databases that occur in the specification. Here, the enumeration variables can be used. For example:

$$\mathsf{Sets} : \mathit{ring}(\mathit{Honest}),\ \mathit{db}(s, A, \mathit{Status});$$

defines that every honest agent Honest has its own keyring $\mathit{ring}(\mathit{Honest})$, which may be for instance a set of public keys, and the server s has a database for each agent A and each Status, each of which may again be a set of public-keys. Thus, this example specification uses $N = 10$ sets.

One can further use the enumeration variables as abbreviations in rules. First, we may use universal quantification of enumeration variables in negative set conditions, e.g.

$$\forall A, \mathit{Status}.\mathit{PK} \notin \mathit{db}(s, A, \mathit{Status})$$

to mean that PK cannot occur in any of the sets covered by expanding all values of the enumeration variables, so this example expands to 8 negative set conditions.

Second, we can parametrize an entire rule over enumeration variables. We may write for instance $\lambda A.\ \Rightarrow \mathsf{iknows}(A)$ to denote that the intruder knows every agent name. We write λ to avoid confusion with quantification: in fact, the meaning of $\lambda X.r$ is the set of rules $\{r[X \mapsto v] \mid v \in V\}$ where V is the enumeration declared for X.

With this syntactic sugar, it is easy to generalize our example specification for any number of honest and dishonest users and servers, namely by replacing the constants by enumeration variables and enumerating the desired set of agents there [54]. The "unrolling" of this sugar is not always

efficient and we plan as future work to investigate strategies for avoiding that in the translation.

2.1.4 Inconsistent Rules

We exclude rules that are "inconsistent" in a certain sense (although their semantics is well-defined):

Definition 4 *A rule* $r = LF \cdot S_+ \cdot S_- \Rightarrow\!\![F]\!\!\Rightarrow RF \cdot RS$ *is called* inconsistent, *if any of the following holds:*

- $t \in M$ *occurs in* S_+ *and* $t \notin M$ *occurs in* S_-, *or*
- $s \in M$ *occurs in* $S_+ \setminus RF$ *and* $t \in M$ *occurs in* RF, *and the rule allows for an instantiation* σ *with* $s\sigma = t\sigma$.

For the remainder, we consider only consistent rules.

The first kind of inconsistent rule is simply never applicable. For the second kind, we get the contradiction only under a particular instantiation, namely when $s\sigma = t\sigma$, because the rule says that the constraint $s\sigma \in M$ should be removed and $t\sigma \in M$ should be added or kept. (The semantics tells us that here the positive constraint to keep $t\sigma \in M$ wins.)

Note that all rules of our running example are consistent; for instance in the second rule, the instantiation $PK\sigma = NPK\sigma$ is not possible because NPK is fresh, and in the third rule such a substitution is also ruled out by the left-hand side constraints $PK \in db(s, a, valid)$ and $NPK \notin db(s, a, valid)$. In fact, the notion that a rule allows for the instantiation $s\sigma = t\sigma$ is purely syntactical (i.e. independent of the actually reachable states).

There are two reasons to exclude inconsistent rules. First, they often result from a specification mistake, i.e. they do not reflect what the user actually wanted to model. Second, the soundness proof of our abstractions below is more complex when allowing the second kind of inconsistent rules.

2.2 Set-Based Abstraction

The core idea of set-based abstraction is the following: we abstract the fresh data according to its membership in the used sets. For instance, if we have three sets s_1, s_2, and s_3, we may abstract all elements that are contained in s_1 but not in s_2 and s_3 into one equivalence class denoted $val(1, 0, 0)$.

In our running example, we have the sets $s_1 = ring(a)$, $s_2 = db(s, a, valid)$, and $s_3 = db(s, a, revoked)$. Thus let $val(1, 0, 0)$ represent the class of all public keys that the user a has created but that are not (yet) registered with the

server s as valid or revoked. The abstract model thus does not distinguish between several different keys that have the same status in terms of set-membership.

The standard way to express the abstract model by Horn clauses in previous approaches does not work with this abstraction. In particular, when the set membership of a constant changes from the abstract value a to the abstract value a', then for every derivable fact $f[a]$ that contains a also $f[a']$ is derivable. This requires an extension with a new kind of rule that can exactly express $f[a] \implies f[a']$ for every context $f[\cdot]$ and which we formalize below. Note that this kind of rule is different from an algebraic equation like $a \approx a'$, because $f[a']$ does not necessarily imply $f[a]$; moreover, it is different from a rewrite rule, because $f[a]$ is not *replaced* by $f[a']$ but both $f[a]$ and $f[a']$ hold.

2.2.1 Definition of the Abstraction

Definition 5 *Consider a set of rules that uses the ground terms $s_1, \ldots, s_N$ in set conditions $t \in s_i$ and $t \notin s_i$ (including the choice of a total order on the s_i). For a state S, we define the function abs_S that maps from $\mathfrak{A}$ to $val(\mathbb{B}^n)$ as follows: $abs_S(c) = val(b_1, \ldots, b_N)$ with b_i true iff $(c \in s_i) \in S$. This induces an equivalence relation (parametrized by a state S) on $\mathfrak{A}$: define $c \equiv_S c'$ iff $abs_S(c) = abs_S(c')$.*

It is indeed unusual that an abstract interpretation depends on states and can change from state to state. This reflects exactly why the databases we want to model do not exactly fit into the standard abstraction approach of protocol verification: the abstract model does not have a notion of states any more. We will see below (in subsection 2.3) how to overcome this problem and define a state-independent abstraction function.

2.2.2 Term Implication Rules

We now introduce the form of rule that allows us to deal with abstractions with the changing set-membership of constants.

Definition 6 *A* term implication rule *has the form*

$$\frac{P_1 \quad \ldots \quad P_n}{s \twoheadrightarrow t}$$

where the P_i are predicates (i.e. facts) and $vars(t) \cup vars(s) \subseteq \bigcup_{i=1}^{n} vars(P_i)$. An implication rule *is either a term implication rule or a Horn clause. We*

often write $A \to C$ *instead of* $\frac{A}{C}$. *We may also write* $A \to C_1 \cdot \ldots \cdot C_n$ *as an abbreviation for the set of rules* $\{ A \to C_i \mid 1 \leq i \leq n \}$.

For implication rules, we define a function that, given a set Γ *of facts, yields all facts that can be derived from* Γ *by one rule application:*

$$\begin{aligned}
\left[\!\!\left[\frac{\phi_1 \quad \ldots \quad \phi_n}{\phi}\right]\!\!\right](\Gamma) &= \{\phi\sigma \mid \phi_1\sigma \in \Gamma \wedge \ldots \wedge \phi_n\sigma \in \Gamma\} \\
\left[\!\!\left[\frac{\phi_1 \quad \ldots \quad \phi_n}{s \twoheadrightarrow t}\right]\!\!\right](\Gamma) &= \{C[t\sigma] \mid \; C[s\sigma] \in \Gamma \wedge \phi_1\sigma \in \Gamma \\
&\qquad\qquad \wedge \ldots \wedge \phi_n\sigma \in \Gamma \quad \}
\end{aligned}$$

Here, $C[\cdot]$ *is a context, i.e. a "term with a hole", and* $C[t]$ *means filling the hole with term* t. *The* least fixed-point *of a set of implication rules* R, *denoted* $LFP(R)$ *is defined as the least set* Γ *that is closed under* $[\![r]\!]$ *for each* $r \in R$.

2.2.3 Translation to Abstract Rules

We now translate the standard transition rules (that work on the real sets) to implication rules of an abstract model (that work on the abstract encoding of set membership). We show in subsection 2.3 that this abstraction is sound.

Definition 7 *Consider a transition rule*

$$r = LF \cdot S_+ \cdot S_- \; \text{=}[F]\Rightarrow \; RF \cdot RS$$

Let $\mathcal{T}_{\mathfrak{A}}(r)$ *be the symbols from* $\mathcal{T}_{\mathfrak{A}}$ *that occur in* r. *We define for each* $t \in \mathcal{T}_{\mathfrak{A}}(r)$ *and for each* $1 \leq i \leq N$:

$$L_i(t) = \begin{cases} 1 & \text{if } t \in s_i \text{ occurs in } S_+ \\ 0 & \text{if } t \notin s_i \text{ occurs in } S_- \\ X_{t,i} & \text{otherwise} \end{cases}$$

$$R_i(t) = \begin{cases} 1 & \text{if } t \in s_i \text{ occurs in } RS \\ X_{t,i} & \text{otherwise, if } L_i(t) = X_{t,i} \text{ and } t \notin F \\ 0 & \text{otherwise} \end{cases}$$

Here, let $X_{t,i} :: \mathbb{B}$ *be variables that do not occur in* r. *Let*

$$\begin{aligned} L(t) &= val(L_1(t), \ldots, L_N(t)) \\ R(t) &= val(R_1(t), \ldots, R_N(t)) \; . \end{aligned}$$

The abstraction $\overline{r}$ of the rule r *is defined as:*

$$\overline{r} = LF\lambda \to RF\rho \cdot C$$

for the following substitutions λ *and* ρ *and term implications* C:

$=[PK]\Rightarrow$ $\mathsf{iknows}(PK) \cdot PK \in ring(a) \cdot PK \in db(s,a,valid)$	$\rightarrow$ $\mathsf{iknows}(val(1,1,0))$
$\mathsf{iknows}(PK) \cdot PK \in ring(a)$ $=[NPK]\Rightarrow$ $NPK \in ring(a) \cdot$ $\mathsf{iknows}(\mathsf{sign}_{\mathsf{inv}(PK)}(new,a,NPK))$	$\mathsf{iknows}(val(1,X_1,X_2))$ $\rightarrow$ $val(1,X_1,X_2) \twoheadrightarrow val(0,X_1,X_2) \cdot$ $\mathsf{iknows}(\mathsf{sign}_{\mathsf{inv}(val(0,X_1,X_2))}(new,a,val(1,0,0)))$
$\mathsf{iknows}(\mathsf{sign}_{\mathsf{inv}(PK)}(new,a,NPK)) \cdot$ $PK \in db(s,a,valid) \cdot NPK \notin db(s,a,valid) \cdot$ $NPK \notin db(s,a,revoked)$ $\Rightarrow$ $PK \in db(s,a,revoked) \cdot$ $NPK \in db(s,a,valid) \cdot$ $\mathsf{iknows}(\mathsf{inv}(PK))$	$\mathsf{iknows}(\mathsf{sign}_{\mathsf{inv}(val(X_1,1,X_2))}(new,a,val(X_3,0,0)))$ $\rightarrow$ $val(X_1,1,X_2) \twoheadrightarrow val(X_1,0,1) \cdot$ $val(X_3,0,0) \twoheadrightarrow val(X_3,1,0) \cdot$ $\mathsf{iknows}(\mathsf{inv}(val(X_1,0,1)))$
$\mathsf{iknows}(\mathsf{inv}(PK)) \cdot PK \in db(s,a,valid)$ $\Rightarrow$ $attack$	$\mathsf{iknows}(\mathsf{inv}(val(X_1,1,X_2)))$ $\rightarrow$ $attack$

Figure 1: The transition rules of the running example (LHS) and their abstraction (RHS).

- $\lambda = [t \mapsto L(t) \mid t \in \mathcal{T}_{\mathfrak{A}}(r)]$
- $\rho = [t \mapsto R(t) \mid t \in \mathcal{T}_{\mathfrak{A}}(r)]$
- $C = \{t\lambda \twoheadrightarrow t\rho \mid t \in \mathcal{T}_{\mathfrak{A}}(r) \setminus F\}$

2.2.4 The Example

Figure 1 shows the translation of our running example. Thanks to the abstraction, it is straightforward to convince oneself that *attack* is unreachable, as this requires the fact $\mathsf{iknows}(\mathsf{inv}(val(X_1,1,X_2)))$ (i.e. a valid key) whereas the only rule that gives the intruder a private key has the incompatible set membership $(X_1,0,1)$ (i.e. a revoked key) and there is no term implication rule that could turn a revoked key into a valid one. Let

$$\begin{aligned} SK &= \{val(0,0,0), val(0,1,0), val(0,0,1)\} \\ K &= SK \cup \{val(1,0,0), val(1,1,0)\} \end{aligned}$$

K is the set of all public keys that occur in some fact. (The other three bearable keys $val(0,1,1)$ and $val(1,1,1)$ and $val(1,0,1)$ do never occur.) The

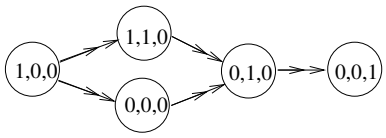


Figure 2: The key life-cycle as formalized by the term implications.

subset SK contains those keys that can ever occur as the signing key in a signature.

The fixed-point is $\Gamma = \{\mathsf{iknows}(m) \mid m \in M\}$ where

$$M = \begin{array}{l} \mathcal{DY}(K \cup \{\mathsf{sign}_{\mathsf{inv}(sk)}(a, new, k) \mid sk \in SK, k \in K\} \\ \cup\{\mathsf{inv}(val(0,0,1))\}) \end{array}$$

and $\mathcal{DY}(\cdot)$ denotes the closure under protocol-independent intruder deduction rules (like encryption). In particular, only the private keys of revoked, invalid keys get known to the intruder, and *attack* is not in Γ.

We note that the concrete term implications $s \twoheadrightarrow t$ which get activated in Γ, displayed in Figure 2, represent exactly the life-cycle of keys.

2.3 Soundness

For verification, the crucial property of our abstraction is that if the concrete model has an attack, then so has the abstract model. If this holds, then verification of the abstract model implies verification of the concrete model.We take a detour over some intermediate models which greatly simplifies the actual proof of correctness.

The labeled concrete model The first idea is to *label* all symbols of $\mathcal{T}_{\mathfrak{A}}$ in the concrete model with the corresponding abstract terms according to Definition 5. Being merely an annotation, this does not change the model.

Definition 8 *The* labeled concrete model *is defined as the following modification rules of the concrete model: every $t \in \mathcal{T}_{\mathfrak{A}}$ on the LHS (RHS) of a rule is labeled with $L(t)$ ($R(T)$) (cf. Definition 7). We denote the labeling of term t with label l by $t@l$. Moreover, for each $t \in \mathcal{T}_{\mathfrak{A}}$ that occurs on both sides, we add the* label modification *$t@L(t) \mapsto t@R(t)$. This label modification is applied as a replacement on the successor state: let $r' = r \cdot (t@l \mapsto t@l')$ the augmentation of r with the label modification. We then define r' transitions based on r transitions as follows: if $S \Rightarrow_r S'$ under match σ then $S \Rightarrow_{r'} S'\tau$ where τ is the replacement of all occurrences of $t\sigma$ (for any label) with $t\sigma$ labeled by l'.*

As an example, the second rule of our running example looks as follows in the labeled model:

$$\begin{array}{l}\mathsf{iknows}(PK@(1,X_1,X_2)) \cdot PK@(1,X_1,X_2) \in ring(a)\\ =\!\![NPK@(1,0,0)]\!\Rightarrow\\ NPK@(1,0,0) \in ring(a)\\ \mathsf{iknows}(\mathsf{sign}_{\mathsf{inv}(PK@(0,X_1,X_2))}(new,a,NPK@(1,0,0))) \cdot\\ PK@(1,X_1,X_2) \mapsto PK@(0,X_1,X_2)\end{array}$$

Lemma 1 *In the labeled model, in every state S, every occurrence of an abstractable constant c is labeled with $l = (b_1, \ldots, b_N)$ such that b_i is true iff the set condition $c \in s_i$ is contained in S.*

Proof. We show this by induction over reachability. It trivially holds for the initial state. For transitions, suppose the property holds in state S, and $S \rightarrow_{r'} S'$ for some labeled rule r' and let σ be the rule match. Consider any $c \in Abs$ that occurs in S' with label $(b_1, \ldots, b_N)$. We show for every $1 \leq i \leq N$: b_i is true iff $c \in s_i$ occurs in S'. We distinguish the following cases:

- c does not occur in S, so it was freshly created by the transition to S'. Thus there is a variable X in the fresh variables of r' such that $X\sigma = c$. By definition, X (and thus every occurrence of c in S') is labeled with $b_i = R_i(X)$ which is true iff $X_i \in s_i$ is contained in the right-hand side of r', which is the case iff $c \in s_i$ is contained in S'.

- c occurs in S, and for no abstractable variable X in r' it holds that $c = X\sigma$. Then c is simply not touched by the transition and has the same label and set memberships in both states.

- c occurs in S, and for some abstractable variable X in r', we have $X\sigma = c$. (Note there may be other variables Y with $Y\sigma = c$.) We further distinguish:

  - $X \in s_i$ occurs in S_+. Then $c \in s_i$ occurs in S and there is no variable Y such that both $Y\sigma = c$ and $Y \notin s_i$ occurs in S_- (otherwise r' would not have been applicable to S under σ). If for any variable Y with $Y\sigma = c$, $Y \in s_i$ occurs in RS, then also $X \in s_i$ must occur in RS otherwise the rule is not consistent (cf. Definition 4). Thus $R_i(X)$ and b_i is true and $c \in s_i$ is contained in S'. Otherwise, if for no variable Y with $Y\sigma = c$, $Y \in s_i$ is contained in RS, then by the definition there is a label change for X in r', namely changing at least the ith position from true to false. Then $c \in s_i$ is not in S' and b_i is false.

  - $X \notin s_i$ occurs in S_-. Then $c \in s_i$ does not occur in S, and there is no variable Y with $Y\sigma = c$ and $Y \in s_i$ in S_+. Suppose for any Y with $Y\sigma = c$, $Y \in s_i$ occurs in RS, then also $X \in s_i$ in RS (otherwise the rule were again inconsistent). Thus there is a label change in the ith position from false to true and b_i is true and $c \in s_i$ is contained in S'. Otherwise, X is labeled on both sides with false for the ith component, and b_i is false, and $c \in s_i$ does not occur in S'.
  - Neither $X \in s_i$ occurs in S_+ nor $X \notin s_i$ occurs in S_-. If $X \in s_i$ occurs in the RS, then we have a label change in the ith position of the labeling of X, namely from arbitrary X_i to 1. Thus b_i is 1 and $c \in s_i$ occurs in S'. Otherwise, if $X \in s_i$ does not occur in RS, then X is not involved in any set conditions. Then either c stays with the same label and set membership in the transition from S to S', or there is another variable Y with $Y\sigma = c$ and any of the above cases can be applied with Y in the role of X.

- c occurs in S but there is no abstractable variable X in r' such that $X\sigma = c$. Then there is no change of set memberships of c and no label change and the property remains that the labeling is correct.

□

Labeled concrete model without set conditions The labels are thus a correct alternative representation of the set conditions, and as a second step we now "upgrade" the labels from a mere annotation to a part of term structure, i.e. considering @ as a binary (infix) function symbol. Then, upon rule matching the label does matter. In turn, we can remove the set conditions from our model completely, because we can always reconstruct the set memberships from the labels (thanks to persistence and rule form, no abstractable constants can get lost on a transition) and the set conditions on the left-hand side of a rule are correctly handled by the label matching. In this *labeled model without set conditions*, the second rule of our running example is:

$$\begin{array}{l}\mathsf{iknows}(PK@(1, X_1, X_2))\\ =[NPK@(1,0,0)]\Rightarrow\\ \mathsf{iknows}(\mathsf{sign}_{\mathsf{inv}(PK@(0,X_1,X_2))}(new, a, NPK@(1,0,0))) \cdot\\ PK@(1, X_1, X_2) \mapsto PK@(0, X_1, X_2)\end{array}$$

Note how close this rule is to the abstract model, while still being a state transition rule. It is immediate from Lemma 1 that this changes the model only in terms of representation:

Lemma 2 *The labeled model and the labeled model without set conditions have the same set of reachable states modulo the representation of set conditions in labels.*

The abstraction All the previous steps were only changing the representation of the model, but besides that the models are all equivalent. Now we finally come to the actual abstraction step that transforms the model into an abstract over-approximation.

We define a representation function η that maps terms and facts of the concrete model to ones of the abstract model:

Definition 9

$$\begin{aligned} \eta(t@(b_1,\ldots,b_N)) &= val(b_1,\ldots,b_N) \text{ for } t \in \mathcal{T}_{\mathfrak{A}} \\ \eta(f(t_1,\ldots,t_n)) &= f(\eta(t_1),\ldots,\eta(t_n)) \\ &\text{for any function or fact symbol } f \text{ of arity } n \end{aligned}$$

We show that the abstract rules allow for the derivation of the abstract representation of every reachable fact f of the concrete model:

Lemma 3 *Let R be a rule set in AIF, R' be the corresponding rule set in the labeled model without set conditions of R, f be a fact in a reachable state of R' (i.e. $\emptyset \to_{R'}^* S$ and $f \in S$ for some S). Let $\overline{R}$ be the translation into Horn clauses of the rules R according to Definition 7, and $\Gamma = LFP(\overline{R})$. Then $\eta(f) \in \Gamma$.*

Proof. Again we show this by induction over reachability. The initial state $\emptyset$ is clear. Let now S be any reachable state and $\eta(f) \in \Gamma$ for every $f \in S$. We show that for every S' that is reached by one rule application and every $f \in S'$ also $\eta(f) \in \Gamma$.

Let the considered rule be

$$r = LF \,\text{=}[F]\Rightarrow\, RF \cdot LM$$

where LM are the label modifications (see Definition 8)—being part of the labeled model without set conditions there are no set conditions in the rule. By our constructions, the Horn clauses $\overline{R}$ contain a similar rule, namely

$$\overline{r} = \eta(LF) \to \eta(RF) \cdot \eta(LM)$$

where we extend η to sets of facts as expected. The extension of η to label modifications (and sets thereof in $\eta(LM)$) is also straightforward:

$$\eta(t@l \mapsto t@l') = val(l) \twoheadrightarrow val(l')$$

Let now σ be the corresponding substitution for $S \rightarrow_r S'$. Then $LF\sigma \subseteq S$ and thus $\eta(LF\sigma) \subseteq \eta(S)$. Thus the Horn clause $\overline{r}$ is applicable and therefore $\eta(RF\sigma) \subseteq \Gamma$. It remains only to show that all the modifications of facts by the label modification rule are also contained in Γ.

To that end, consider any fact $f[c@l] \in (S \cup RF)\sigma$ that has exactly one occurrence of $c@l$ and LM contains the rule $t@l \mapsto t@l'$ for some t with $t\sigma = c$. Since $l \twoheadrightarrow l'$ is part of the term implication of $\overline{r}$ and since we have $\eta(f[c@l]) \in \Gamma$, we also have $\eta(f[c@l']) \in \Gamma$. If there is more than one occurrence of an abstractable constant in a fact that is affected by a label modification, then we can repeatedly apply this argument. Note that the term implication of the (generalized) Horn clauses only replace one occurrence at a time. The reason is that from the label l we cannot be sure that all its occurrences correspond to the same constant $c@l$ in the concrete model, so replacement of only part of the labels is included.

We have thus shown that all the facts in S' are also contained in Γ, modulo the representation function η. □

From Lemmata 2 and 3 immediately follows that the over-approximation is sound:

Theorem 1 *Given an AIF specification with rules R. If an attack state is reachable with R, then attack $\in LFP(\overline{R})$.*

2.4 Encoding Term Implication

We show how the term implication rules that we have introduced can be encoded into Horn clauses. Intuitively, the problem is that the rule $s \twoheadrightarrow t$ expresses $C[s] \implies C[t]$ for any context C, and thus summarizes an infinite number of Horn clauses. However, this infinite enumeration can be avoided by limiting ourselves to ones that can be instantiated to a derivable fact. This can be done using a new constant symbol ϵ and two new binary fact symbols *occurs* and *implies* (i.e. these symbols do not occur in the given specification). $occurs(p, t)$ expresses that t is a subterm of some fact that holds, and either

- p is ϵ, then t is a direct subterm of a fact that holds, or
- p is also a subterm of a fact that holds, and t is a direct subterm of p.

Further, $implies(s, t)$ represents a rule of the form $s \twoheadrightarrow t$. For every n-ary fact symbol f (not including *occurs* and *implies*), every m-ary operator g,

every $1 \leq i \leq n$ and every $1 \leq j \leq m$, we have the following Horn clauses:

$$
\begin{array}{l}
f(x_1, \ldots, x_n) \rightarrow \mathit{occurs}(\epsilon, x_i) \\
\mathit{occurs}(x, g(y_1, \ldots, y_m)) \rightarrow \mathit{occurs}(g(y_1, \ldots, y_m), y_j) \\
\mathit{occurs}(g(x_1, \ldots, x_m), x_j) \cdot \mathit{implies}(x_j, y) \\
\quad \rightarrow \mathit{implies}(g(x_1, \ldots, x_m), g(x_1, \ldots, x_{j-1}, y, x_{j+1}, \ldots, x_m)) \\
f(x_1, \ldots, x_n) \cdot \mathit{implies}(x_i, y) \\
\quad \rightarrow f(x_1, \ldots, x_{i-1}, y, x_{i+1}, \ldots, x_n)
\end{array}
$$

Let us call these Horn clauses R_0. Consider an arbitrary set of Horn clauses R_h and term implication rules R_t. Call R'_t the Horn clauses that are obtained from R_t by replacing the consequence $s \twoheadrightarrow t$ by the fact $\mathit{implies}(s, t)$.

Theorem 2 $\mathit{LFP}(R_h \cup R_t) = \mathit{LFP}(R_0 \cup R_h \cup R'_t) \setminus \{\mathit{implies}(\cdot,\cdot), \mathit{occurs}(\cdot,\cdot)\}$

Proof. Let $\Gamma = \mathit{LFP}(R_h \cup R_t)$ and $\Gamma' = \mathit{LFP}(R_0 \cup R_h \cup R'_t)$ and $\Gamma'' = \Gamma' \setminus \{\mathit{implies}(\cdot,\cdot), \mathit{occurs}(\cdot,\cdot)\}$.

Soundness, i.e. $\Gamma'' \subseteq \Gamma$: $\mathit{occurs}(\cdot, t) \in \Gamma'$ only holds for subterms t of facts in Γ and $\mathit{implies}(t_1, t_2)$ only holds if for any fact $C[t_1] \in \Gamma$ also $C[t_2] \in \Gamma$ holds. As a consequence, the last rule schema of R_0 can only give facts that are in Γ.

Completeness, i.e. $\Gamma \subseteq \Gamma'$: Suppose $f \in \Gamma \setminus \Gamma'$, and suppose f is the "shortest" counter-example, i.e. it can be derived with one rule application of R_t from Γ' (it cannot be a rule from R_h since Γ' is closed under R_h). Let $\phi_1, \ldots, \phi_n \rightarrow s \twoheadrightarrow t$ be that rule, σ the substitution under which it is applied and thus $f = C[t\sigma]$ for some context $C[\cdot]$. By the assumption of shortest counter-example, $\phi_i\sigma \in \Gamma'$ and $C[s\sigma] \in \Gamma'$. Thus we also have $\mathit{implies}(s\sigma, t\sigma) \in \Gamma'$. Moreover, $\mathit{occurs}(\cdot, u) \in \Gamma'$ for all subterms of $C[s\sigma]$ and by that we have the $\mathit{implies}(\cdot,\cdot)$ over corresponding subterms of $C[s\sigma]$ and $C[t\sigma]$. Thus, finally, $C[t\sigma] \in \Gamma'$. □

2.5 Decidability

It is straightforward to adapt, to our AIF formalism, the classical proof of [41] that protocol verification is undecidable. This is because this proof relies only on intruder deduction rules that can be applied without any bounds. Moreover, since it does not even use fresh constants, the proof also applies to the abstracted model of an AIF specification. Thus, the security of AIF specification is undecidable both in the concrete and abstract model.

Let us consider the restriction that all rule variables can be instantiated only with variables of a given depth. Such a bounding of substitutions is without loss of attacks in a typed model that can be justified for a large

class of protocols by tagging [45]. For the abstract model, decidability is now obvious, because this makes the set of derivable terms finite. For the concrete model, however, we now show that verification is undecidable even when bounding the message depth. [40] shows this for verification in a standard multi-set rewriting approach, but their proof cannot be carried over to AIF immediately because AIF only supports persistent facts and membership conditions for a fixed number of sets. We show that it is expressive enough, however, to simulate Turing machines and thus obtain the following decidability results:

Theorem 3 *Reachability of the attack fact is undecidable both in the concrete and in the abstract model (even when using no sets and fresh data). With a depth restriction on substitutions, the abstract model is decidable, while the concrete model remains undecidable.*

PROOF. The idea for encoding Turing machines into message-bounded AIF is that every position of the tape is modeled by a fresh constant, and the symbol is carried by set containment. In an initialization phase, we generate an arbitrary long but finite tape—the length is chosen non-deterministically:[2]

$$\Rightarrow westend(c_0)$$

$$westend(c_0) \cdot c_0 \notin initializing$$
$$=[X]\Rightarrow c_0 \in initializing \cdot succ(c_0, X) \cdot X \in current$$

$$c_0 \in initializing \cdot X \in current$$
$$=[Y]\Rightarrow succ(X, Y) \cdot Y \in current \cdot c_0 \in initializing$$

$$c_0 \in initializing \cdot X \in current \; =[Y]\Rightarrow$$
$$succ(X, Y) \cdot eastend(Y) \cdot c_0 \in current \cdot$$
$$c_0 \in q_0 \cdot c_0 \in computing$$

where q_0 is the initial state of the machine. For every machine transition $(q, s) \rightarrow (q', s', L)$ the rule

$$c_0 \in computing \cdot X \in current \cdot X \in q \cdot X \in s \cdot succ(Y, X)$$
$$\Rightarrow c_0 \in computing \cdot Y \in current \cdot Y \in q' \cdot X \in s'$$

The rules for moving right and neutral are similar. Additionally, when the machine reaches the eastend of the tape (which only exists in our model), we go to a sink state of the model from which no further progress can be made:

$$c_0 \in computing \cdot X \in current \cdot X \in q \cdot X \in s \cdot eastend(X)$$
$$\Rightarrow c_0 \in stuck$$

[2]The finiteness is not a restriction as we use a special sink state when reaching the eastend of the tape.

Note that one can easily also encode an initial value of the tape. The Turing machine can reach a certain state q, if the concrete model has a reachable state that contains $c \in q$ for some value c. This can, of course, also be formalized by an attack rule. For this model, a depth bound for variables of 1 (i.e. variables can only be substituted by constants) is no restriction. As reachability of states for a Turing machine is undecidable, so is the reachability of an attack state in the depth bounded concrete model. □

2.6 Experimental Results

Problem	Agents	Result	SPASS Time	ProVerif Time
Key-server example	a, i, s	safe	1s	0s
	a, b, c, i, s	safe	37s	0s
SEVECOM (one key)	$hsm, auth, i$	safe	12s	0s
(both keys)	$hsm, auth, i$	unsafe	0s	timeout
ASW	a, i, s	safe	3hrs	6min
TLS (simplified)	a, i	safe	1s	0s
	a, b, i	safe	75s	13s
NSL (w. conf. ch.)	a, b, i	safe	17s	0s
NSPK (w. conf. ch.)	a, b, i	unsafe	0s	0s

Table 1: Experimental results using SPASS and ProVerif

We have implemented the translation from AIF to a set of Horn clauses as described in the previous subsections both for the syntax of the theorem prover SPASS and for the syntax of the protocol verifier ProVerif. This implementation along with a library of AIF specifications is available, including more detailed descriptions of the examples presented here [54].

Recall that above we explicitly said that we want to interpret terms and Horn clauses in the free algebra: terms are interpreted as equal iff they are syntactically equal. For instance, for different constants a and b, $a = b$ is false. The same is not necessarily true in first-order logic: it rather depends on the structure (i.e. universe and interpretation of all functions and relation symbols) in which a formula is interpreted. Thus, there are interpretations in which the formula $a = b$ holds. A formula is *valid*, if it holds in all interpretations (e.g. $a = b \rightarrow b = a$).

The SPASS theorem prover allows us to declare a list of axioms $\phi_1, \ldots, \phi_n$ and a conjecture ϕ. It will then try to prove or disprove that $\phi_1 \wedge \ldots \wedge \phi_n \implies \phi$ is valid. We use as the axioms ϕ_i the Horn clauses that result from the

translation of AIF and as the conjecture ϕ we use simply *attack*. When SPASS returns "proof found", we know that there is indeed an attack (against the abstract model), as that can be derived from the Horn clauses in any interpretation of the symbols, including the free algebra interpretation. When SPASS however returns "completion found", then for at least one interpretation, *attack* cannot be derived. Of course this means, that the attack cannot be derived in the free algebra interpretation (because if it can be derived in the free algebra interpretation, then it can be derived in any interpretation). Thus if SPASS finds a completion, we know the given protocol is secure in the abstract model with the free algebra interpretation [61] and by the soundness also in the concrete model.

The translation to ProVerif is similar, where we may exploit domain-specific optimizations, such as treatment of the intruder-knowledge fact. In general it turns out that ProVerif is faster than SPASS in finding results, see Table 1, which is not surprising as ProVerif is a dedicated, specialized tool. (The exception where ProVerif times out is discussed below.) We have noted the number of agents that were used in each example, and it can be seen that this has a major influence on the run-time. This is of course due to the fact that with the number of agents, also the number N of sets in our model increases and the number of equivalence classes underlying the abstraction is 2^N.

As the first concrete example, we have considered our key-server example, albeit with several honest and dishonest participants. The second example analyzes part of a system for secure vehicle communication from the SEVECOM project [58]. Here, each car has a hardware security module HSM that, amongst others, stores two public root keys of an authority (for verifying messages sent by the authority). The reason for using two root key pairs is that even if one private key is leaked, the authority can still safely update it using the other. We have found some new attacks that were missed in the analysis of [59], because that model does not include the authority (and thus no legal update messages). The attacks are practically limited as they require either several updates within a short period of time or that there is a confusion about which key has been leaked (i.e. the intruder knows one key and the authority updates the other). We have verified the system under the following simple restriction (see [54] for other suggestions to avoid the attacks): we assume that one of the two private keys is never leaked and thus never needs an update, while the other key may be leaked and updated any number of times. Under this restriction, we can verify the following goals: the intruder never finds out private keys (except for ones we give him deliberately), he cannot insert into the HSM any keys he generated himself, and he cannot re-insert old keys. Finally, if we give the intruder both keys, the

resulting trivial attack is found by SPASS immediately while ProVerif times out. The reason seems to be that ProVerif dives into the more complicated derivations enabled by the additional intruder knowledge before finding the attack. We will investigate this behavior further as it occurred several times during our experimentation with this example set.

The largest example, and in fact one of the original motivations for this work, is the contract signing protocol ASW based on optimistic fair-exchange [6]. Again, we restrict our discussion to a short summary of ASW and highlighting some key issues of the formalization in AIF, more details are found in [54]. The idea is that two parties can sign a contract in a fair way, i.e. such that finally either both parties or no party has a valid contract. This requires in general a trusted third party TTP, which for ASW is only needed for resolving disputes. The TTP maintains a database of contracts that it has processed so far, which are either aborted or resolved. A resolve means that the TTP issues a valid contract. Whenever an agent asks the TTP for an abort or resolve, the TTP checks whether the contract in question is already registered as aborted or resolved. If this is not the case, then the request to abort or resolve is granted, otherwise the agent gets the abort token or replacement contract stored in the database.

The protocol is based on nonces to which the exchange is bound. Therefore each agent including the TTP maintains a database of nonces. The database stores for each nonce to which parties it relates, to which contractual text, and the status of the respective transaction. For the TTP, the status is just aborted or revoked, for honest agents the status is the stage in the protocol execution (there are several rounds and exceptions). One of the major difficulties of this case study is that the fair exchange relies on the assumption of resilient channels between agents and the TTP, i.e. the intruder (which may be a dishonest contractual partner) cannot block the communication forever. For this, we use a model where the request from the user and the answer from the TTP happen in a single transition. Roughly speaking, we have three cases for each party asking for an abort (and three similar for resolve requests):

- The party is in a stage of the protocol execution where it can ask for an abort, and the TTP has not previously seen the nonce contained in the abort request, i.e. it was not involved in a resolve or an abort. Then we can go to a state where both the party and the TTP have noted the nonce as aborted.

- The other two rules are similar but for the case that the TTP has already noted the nonce as aborted or as resolved and this result is communicated to the agent.

While in general, the handling of resilient channels cannot be done by such a contraction of several steps into a single one, the model in this case covers all real executions if we assume that no honest party sends several requests at a time and that the TTP processes requests sequentially.

Another challenge are the goals of fair exchange itself, namely when one party has a valid contract, then the other one can eventually obtain one. This is in fact a liveness property and cannot directly be expressed. We use here the fact that every agent who does not obtain a contract will eventually contact the trusted third party and get either an abort or resolve. Thus, it is sufficient to check that we never come to a state where one party has a valid contract and the other one has an abort for that contract; this is a safety property.

Finally, we have also considered some "normal" protocols that do not rely on databases, namely a simplified version of TLS, the famous flawed NSPK and the fixed variant by Lowe (NSL) [49]. The reason is that these protocols are standard examples. Also this demonstrates that we can use databases of nonces or keys as an alternative way to describe the relevant state-information of agents. For NSPK and NSL we use confidential channels instead of public-key encryption.

The experimental results demonstrate that our abstraction approach is feasible for a variety of verification problems of security protocols and web services.

2.7 Concluding Remarks

The abstraction and over-approximation of protocols and web services by a set of Horn clauses is a very successful method in practice [17, 25, 23, 61, 16]. In contrast to classical model-checking approaches, this kind of over-approximation does not suffer from the usual interleaving problems and can verify protocols for an unbounded number of sessions. The technique has however limitations for protocols and web services that are based on databases of keys, contracts, or even access rights, where revocation is possible, so that the set of true facts does not monotonically grow with the transitions.

We present a new way of abstraction in the spirit of the Horn clauses approach that can handle such databases and thus broadens the scope of this abstraction method. The abstraction of data we propose is based on the membership of the data in the databases. The updating of the databases requires also an update of the abstraction of the data which we can declaratively express with a new form of rule we have introduced, the term implication rule. We show how to encode this rule into standard Horn clauses. As a

consequence we can use with ProVerif an existing tool from the abstraction community, and even the general purpose first-order theorem prover SPASS. The SEVECOM and ASW examples show that our method is feasible for modeling complex real-world systems with databases and APIs that, for reasons of their non-monotonic behavior, were previously out of the scope of the standard abstraction-based methods. While the AIF-library is still small, this suggest that our method is practically feasible to tackle exactly what is missing for the verification of more complex cryptographic systems.

[60] considers an abstraction of keys in an API by attributes; this has some similarity with our set-membership abstraction. However, the attributes in [60] are static (i.e. set memberships cannot change).

Our new language AIF gives a convenient way of writing specifications in an un-abstracted form. Still, AIF is too low-level to be used by a protocol or web service designer. We thus plan as part of future work to connect more high-level languages. Also we plan to build a tool with native support for the term-implication rules and for other improvements specific to our approach. Further, the approach is currently limited to a fixed number N of sets; we plan to investigate how we can avoid this limitation. Another interesting question we want to consider is the relation of our approach to two quite different approaches, namely static analysis [21] and type-based analysis [15], which, besides all differences, show some similarities with our approach.

3 Encoding security-policy clauses

A security-sensitive service, e.g. specified in ASLan++, can be decomposed into the *communication* and the *policy* levels: security protocols executed by the service constitute the communication level, while the policy engine that regulates the behaviors of the service constitutes the policy level. The separation between the communication and policy levels is a useful abstraction for better understanding each of these levels.

For deciding reachability in security protocols (i.e. the communication level), it is assumed that the attacker is in direct control of the communication media, i.e. messages are passed through the attacker. The message composition capabilities of the attacker often reflect the Dolev-Yao threat model [39]. For deciding reachability in security-sensitive services, we need to also take the policy engines of services into account. Towards a seamless decision algorithm which accounts for both the communication and policy levels of services, we propose an encoding that shows that (a fragment of) policy level computations of services can be seen as message derivations in the Dolev-Yao attacker model. Then, intuitively, the attacker and all the services would be equipped with the reasoning power of the Dolev-Yao model, which is well understood and comes with decision algorithms for reachability. The encoding would thus benefit us in two ways: (1) it simplifies the decidability proofs as it builds upon the decidable Dolev-Yao threat model, and (2) it allows us to use (with minor modifications) the existing tools which have been originally developed for verifying security protocols in order to decide reachability in security-sensitive services.

Note that decision algorithms for correctness of security protocols and policy engines have been mostly developed in isolation. For instance, it has been shown that the secrecy problem is decidable for security protocols with a bounded number of sessions [56, 51]. For these results, the local computational power of the processes is limited to pattern matching, hence not fully accounting for authorization policies of the participants. Likewise, (un)decidability results for the safety problem in the HRU access control matrix model, and authorization logics such as [37, 14, 43] abstract away communication level events and their effects on policy level decisions. In contrast, we aim towards a decision algorithm for reachability which takes the communication and policy levels into account, and also covers the interface between them.

In this section, we focus on a fragment of policies in which the policies of services are expressed in terms of *trust application* and *trust delegation* rules à la DKAL [43, 44], and can also express typical RBAC models with role hierarchy. The trust application and trust delegation rules are the core

of many authorization logics [37, 14, 43, 44]. Informally, they state that

Trust application: If Alice trusts Bob on statement f, and Bob says f, then Alice believes f holds.

Trust delegation: If Alice trusts Bob on statement f, then she trusts Bob also on delegating the right to state f to others, e.g. to Charley.

Trust delegation is an important mechanism to provide resilience and flexibility in distributed systems. In practice, however, for a given application, trust delegation may, or may not, be allowed. The results presented here can be adapted to exclude (transitive) trust delegation, if desired.

3.1 Preliminaries

A *signature* is a tuple $(\Sigma, \mathcal{V}, \mathcal{P})$, where Σ is a countable set of functions, $\mathcal{V}$ is a countable set of variables, $\mathcal{P}$ is a nonempty finite set of predicates, and these three sets are pairwise disjoint. We use capital letters $A, B, \ldots$ to refer to the elements of $\mathcal{V}$. The free term algebra induced by Σ, with variables $\mathcal{V}$, is denoted $\mathcal{T}_{\Sigma(\mathcal{V})}$. A *message* is an element of $\mathcal{T}_{\Sigma(\emptyset)}$, i.e. a ground term. The set of *atoms* $\mathcal{A}_{\Sigma(\mathcal{V})}$ is defined as $\{p(t_1, \cdots, t_n) \mid p \in \mathcal{P}, t_i \in \mathcal{T}_{\Sigma(\mathcal{V})},$ with arity of $p = n\}$. The total function $var : \mathcal{T}_{\Sigma(\mathcal{V})} \cup \mathcal{A}_{\Sigma(\mathcal{V})} \rightarrow 2^{\mathcal{V}}$ returns the set of variables appearing in a term or an atom. A *fact* is an atom with no variables.

Let s be a finite set of facts, and $\mathcal{I}$ be a finite set of Horn clauses; a Horn clause is of the form $a \leftarrow a_1, \cdots, a_n$, with $n \geq 0$, and $a, a_1, \cdots, a_n$ being atoms. The *closure* of s under $\mathcal{I}$, denoted $\lceil s \rceil^{\mathcal{I}}$, is the smallest set that contains s and moreover $\forall (a \leftarrow a_1, \cdots, a_n) \in \mathcal{I}. \forall \sigma.\ a_1\sigma, \cdots, a_n\sigma \in \lceil s \rceil^{\mathcal{I}} \implies a\sigma \in \lceil s \rceil^{\mathcal{I}}$, where σ is a total (grounding) substitution function for the Horn clause $a \leftarrow a_1, \cdots, a_n$; that is $\sigma : (var(a) \bigcup_{1 \leq i \leq n} var(a_i)) \rightarrow \mathcal{T}_{\Sigma(\emptyset)}$. The existence of $\lceil s \rceil^{\mathcal{I}}$ follows immediately from Knaster-Tarski's fixed point theorem.

The *policy engine* of a service is a pair $(\Omega, \mathcal{I})$, where Ω is a finite set of facts, called the *policy statements*, and $\mathcal{I}$ is a finite set of Horn clauses, called the *intensional knowledge* of the service. A fact f is *derived* in the policy engine $(\Omega, \mathcal{I})$ iff $f \in \lceil \mathcal{I} \rceil^{\Omega}$.

3.2 Policy engines and message terms

We introduce the notion of *infons*, which we borrow from DKAL [43, 44]. Infons are pieces of information, e.g. $can\_read(bob, file12)$ stipulating that Bob can read a certain *file12*. An infon does not admit a truth value, i.e. it is never false or true. Instead, if the policy engine of a service, say Alice,

derives the predicate $knows(can\_read(bob, file12))$, then Alice "knows" that Bob may read this file, and may thus grant him read access to *file12*. Note that "knows" in this context, and also in DKAL, is a predicate symbol and not a modality as in logics of knowledge. In fact, "knows" here is closer to the notion of *belief* rather than *knowledge*, in epistemic terms.

Infons are different from predicates (i.e. policy statements) in that they are constructed by applying *infon constructors* to message terms and other infons. Below, we assume that in any signature $(\Sigma, \mathcal{V}, \mathcal{P})$, the set Σ can be partitioned into Σ_{msg} and Σ_{infon}, so that Σ_{infon} is the set of infon constructor functions, and Σ_{msg} is the set of message constructor functions. The set of infons, referred to as *Infons*, is formally defined as the smallest set satisfying the following property: if $t_1, \cdots, t_n \in \mathcal{T}_{\Sigma_{msg}(\mathcal{V})} \cup Infons$ and $f \in \Sigma_{infon}$ with arity of f being n, then $f(t_1, \cdots, t_n) \in Infons$. Note that $Infons \cap \mathcal{T}_{\Sigma_{msg}(\mathcal{V})} = \emptyset$; in particular messages are not infons.

Below, we assume that for services the policy statements (i.e. the inhabitants of the policy engines) are predicates over *Infons*. That is, the knowledge of a service ranges over pieces of information.

All the signatures $(\Sigma, \mathcal{V}, \mathcal{P})$ appearing in the rest of this section are assumed to satisfy the following properties:

- $\Sigma = \Sigma_{msg} \sqcup \Sigma_{infon}$ where: [3]
  - A finite subset of constants in Σ_{msg}, denoted by *Agents*, represents the set of the names of the participating services. Intuitively, each service has a name; the names of the services are collected in the set *Agents*.
  - Apart from nullary functions (i.e. constants), Σ_{msg} only contains the functions $\{\cdot\}_\cdot$, $\{\!|\cdot|\!\}_\cdot$, $sig(\cdot,\cdot)$, $pk(\cdot)$, $h(\cdot)$, $(\cdot,\cdot)$. These represent respectively asymmetric and symmetric encryption, digital signature, public key constructor [4], hash and pairing functions, interpreted as usual. We may write x, y for the pair (x, y), when confusion is unlikely.
  - Σ_{infon} contains in particular the functions $\theta(\cdot,\cdot)$ and $\sigma(\cdot,\cdot)$. These intuitively stand for "*trusted on*" and "*said*", respectively, with $\theta, \sigma : Agents \times Infons \to Infons$

[3] Disjoint union of two sets is denoted by $\sqcup$.

[4] Intuitively, one (or more) public key is attributed to each element of *Agents*. The public keys are known to everyone – to the attacker in particular. Using the public key of $a \in Agents$ one can encrypt messages for a and can verify the authenticity of the messages signed by a. See Deliverable D2.3 [8] for further explanations.

- $\mathcal{P} = \{\mathcal{K}\}$, with $\mathcal{K}$ being a unary predicate. Intuitively, $\mathcal{K}$ stands for "knows".

Below, we introduce a class of policy engines, called the $\mathbf{A}_1$ fragment. The $\mathbf{A}_1$ fragment is the target of the encoding that we present in § 3.3. The policy engines in $\mathbf{A}_1$ are equipped with two designated infons to model "said" and "trusted on". These infons are respectively denoted by σ and θ, as mentioned above. Intuitively, $\sigma(a, x)$ states that a said x, while $\theta(a, x)$ states that a is trusted on x. Trust delegation and trust application are the only rules which (relate, and) allow reasoning about σ and θ in the $\mathbf{A}_1$ fragment. A policy engine in the $\mathbf{A}_1$ fragment may however have other rules for manipulating infon constructors different from σ and θ. We require that these rules are "acyclic". These notions are formally defined below.

Definition 10 *[Fragment $\mathbf{A}_1$] Fragment $\mathbf{A}_1$, defined over the signature $(\Sigma, \mathcal{V}, \mathcal{P})$, contains all policy engines $(\Omega, \mathcal{I})$ that satisfy the following syntactical conditions:*

- *Ω is a finite set of ground atoms of the form $\mathcal{K}(i)$, with $i \in$ Infons.*
- *$\mathcal{I}$ includes the TA and TD rules, respectively represented by*

$$\mathcal{K}(X) \leftarrow \mathcal{K}(\theta(A, X)), \mathcal{K}(\sigma(A, X)), \text{ and}$$
$$\mathcal{K}(\theta(A, \theta(B, X))) \leftarrow \mathcal{K}(\theta(A, X)).$$

The set of all the other rules in $\mathcal{I}$ constitutes a type-1 theory*, as defined below, and σ and θ do not appear in this set.*

We define type-1 theories in order to extend the policies of services beyond *TA* and *TD*. Intuitively, type-1 theories contain the rules which a policy engine uses for reasoning about infon constructors different from σ and θ. Type-1 theories can, e.g., express typical RBAC systems with role hierarchy.

Definition 11 (Type-1 theories) *A finite set of Horn clauses T, defined over signature $(\Sigma, \mathcal{V}, \mathcal{P})$, with $\Sigma = \Sigma_{msg} \sqcup \Sigma_{infon}$, is called a* type-1 *theory iff*

(a) All clauses in T have the form $p(t) \leftarrow p_1(t_1), \cdots, p_\ell(t_\ell)$, where $p, p_1, \cdots, p_\ell \in \mathcal{P}$, and $t, t_1, \cdots, t_\ell \in$ Infons.

(b) For all $a \leftarrow a_1, \cdots, a_\ell$ in T, $\bigcup_{i \in \{1, \cdots, \ell\}} var(a_i) \subseteq var(a)$.

(c) The infon dependency graph *of T is acyclic. The infon dependency graph of T is a directed graph defined by the pair $(\Sigma_{infon}, Edges)$ with $(f, g) \in Edges$ iff there exists a Horn clause in T using g in its head and f in its body.*

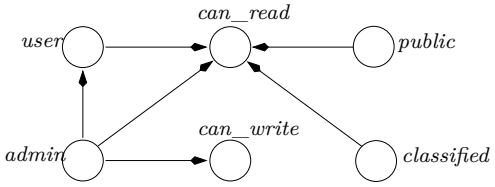


Figure 3: Dependency graph, example 3.1.

We remark that neither *TA* nor *TD* fall into type-1 theories, due to conditions (*b*) and (*c*) in Definition 11, respectively.

Example 3.1 *Consider a file server which implements an RBAC system with two roles,* user *and* admin*. Users may read any* public *file, admins may read any* classified *file, and admins may also write to any file. Admins inherit all the rights attributed to users. The following monadic Horn theory, describing this RBAC system, is indeed a type-*1 *theory.*

$$\begin{array}{lcl}
\mathcal{K}(user(A)) & \leftarrow & \mathcal{K}(admin(A)) \\
\mathcal{K}(can_read(A,F)) & \leftarrow & \mathcal{K}(user(A)), \mathcal{K}(public(F)) \\
\mathcal{K}(can_read(A,F)) & \leftarrow & \mathcal{K}(admin(A)), \mathcal{K}(classified(F)) \\
\mathcal{K}(can_write(A,F)) & \leftarrow & \mathcal{K}(admin(A))
\end{array}$$

Here Σ_{msg} *contains the set of identities of involved services, and names of files, while* $\mathcal{P} = \{\mathcal{K}\}$*, and* $\Sigma_{infon} = \{user, admin, public, classified, can\_read, can\_write\}$ *with obvious arities. The infon dependency graph for this theory, shown in figure 3, is acyclic.*

The following example shows how certain policies on e-health records can be encoded in type-1.

Example 3.2 *In a hospital, each patient is associated with two different sorts of e-health records (e-HR): normal, and emergency. A physician can access all e-HRs of her patients. A nurse can access normal e-HRs of his patients. Nurses can also access emergency e-HRs of any patient, if they belong to the emergency ward "ew". The e-HR server of the hospital manages access to e-HRs. The following type-1 theory formalizes these policies for the e-HR server.*

$$\begin{array}{lcl}
\mathcal{K}(can_access(A, n(P))) & \leftarrow & \mathcal{K}(rel(A,P)), \mathcal{K}(phy(A)) \\
\mathcal{K}(can_access(A, e(P))) & \leftarrow & \mathcal{K}(rel(A,P)), \mathcal{K}(phy(A)) \\
\mathcal{K}(can_access(A, n(P))) & \leftarrow & \mathcal{K}(rel(A,P)), \mathcal{K}(nurse(A)) \\
\mathcal{K}(can_access(A, e(P))) & \leftarrow & \mathcal{K}(ew(A)), \mathcal{K}(nurse(A))
\end{array}$$

Here Σ_{msg} contains the set of identities of physicians, nurses and patients, and two designated functions n and e which return respectively the normal and emergency e-HRs of a given patient. Moreover, $\Sigma_{infon} = \{phy, nurse, can\_access, rel, ew\}$ with obvious arities, and $\mathcal{P} = \{\mathcal{K}\}$. For physician or nurse A and patient P, $rel(A, P)$ denotes that P is a patient of A.

3.3 Encoding policy level computations

Below, we suppress the predicate symbol $\mathcal{K}$ from facts, and work directly with infons; indeed $\mathcal{K}$ is the only predicate symbol in $\mathbf{A}_1$ policy engines. The encoding consists of two functions: ζ which maps infons to $\mathcal{T}_{\Sigma(\mathcal{V})}$, and $\mathcal{E}$ which maps infons to disjunctions of infons (defined below). To simplify the presentation, we start with an initial encoding for trust application only. This encoding is then extended to cover trust delegation and type-1 theories.

We assume the usual capabilities of the Dolev-Yao attacker in composing and decomposing messages. These capabilities are formalized, e.g., in Deliverable D2.3 [8].

3.3.1 TA only.

We recursively define the encoding for infon i:

$$\zeta(i) = \begin{cases} \{\!|\zeta(X), sig(\bar{A}, \zeta(X))|\!\}_{\zeta(\theta(A,X))} & \text{if } i = \sigma(A, X) \\ \theta(\bar{A}, \zeta(X)) & \text{if } i = \theta(A, X) \\ i & \text{otherwise} \end{cases}$$

where $\bar{\cdot} : Agents \rightarrow \overline{Agents}$ is a bijection which associates a unique name to each element of $Agents$. Elements of $\overline{Agents}$ belong to $\mathcal{T}_{\Sigma_{msg}(\emptyset)}$, and are defined solely for the encoding function ζ, i.e. they do not appear in the policy engines.

Here, the encoding of infon $\sigma(a, x)$ is the cipher-text $\{\!|x, sig(\bar{a}, x)|\!\}_{\theta(\bar{a},x)}$ from which the infon x (i.e. what service a said) can be obtained using the decryption rule (of the Dolev-Yao model) only if the key $\theta(\bar{a}, x)$ (i.e. a is trusted on x) is obtained first. This indicates that if TA is applicable on a set of facts at the policy level, then the symmetric decryption rule of the Dolev-Yao inference system, i.e.

$$\frac{\{\!|X|\!\}_K \quad K}{X} \quad Sdec$$

is applicable to the terms resulting from the encoding.

Intuitively, the role of the signature *sig* is to ensure that terms of the form $\{\!|x, sig(\bar{a}, x)|\!\}_{\theta(\bar{a},x)}$ can be constructed using the Dolev-Yao rules only if a corresponding $\sigma(a, x)$ can be derived in the policy level. This is based on the assumption that the attacker does not know the private keys of $\bar{a}$ for any $a \in Agents$.

3.3.2 TA, TD and type-1 theories

In order to include *TD* and type-1 theories in the encoding, we define an *expansion* function $\mathcal{E}$ that for any atom returns a "guard". A guard is of the form $g_1 \vee \cdots \vee g_n$, where $g_1, \cdots, g_n$ are finite sets of atoms. We write $g_i \in_\vee g$ if $g = g_1 \vee \cdots \vee g_i \vee \cdots \vee g_n$, with $n \geq 1$. Intuitively, a guard g is interpreted as the "disjunction" of the "conjunctions" of the atoms in each $g_i \in_\vee g$.

We motivate the expansion function via a simple example. Suppose the fact $\theta(a, i)$ is present in the policy engine of a service, and the query $\theta(a, \theta(b, i))$ is to be evaluated. The *TD* rule implies that the query can be derived, while there is no corresponding inference tree (in the Dolev-Yao model) for $\zeta(\theta(a, \theta(b, i)))$, given $\zeta(\theta(a, i))$. The set of infons which yield $\theta(a, \theta(b, i))$ via applying only the *TD* rule is however finite. This finite set of infons can be seen as a guard, namely $\{\theta(a, \theta(b, i))\} \vee \{\theta(a, i)\}$. The fact that $\theta(a, i)$ yields $\theta(a, \theta(b, i))$ in the policy engine can then be reflected in the Dolev-Yao model as: either $\zeta(\theta(a, \theta(b, i)))$ or $\zeta(\theta(a, i))$ (or both) are obtained from $\zeta(\theta(a, i))$.

The expansion $\mathcal{E}_P(Q, i)$ is defined for finite sets of Horn clauses P and Q, and infon i:

$$\begin{array}{l}\mathcal{E}_P(\emptyset, i) = \{i\} \\ \mathcal{E}_P(\{r \leftarrow r_1, \ldots, r_\ell\} \sqcup Q', i) = \\ \qquad \begin{cases} \mathcal{E}_P(Q', i) \vee \big(\mathcal{E}_P(P, r_1\rho) \cup \cdots \cup \mathcal{E}_P(P, r_\ell\rho)\big) & \text{if } i = r\rho \\ \mathcal{E}_P(Q', i) & \text{if } \neg\exists\rho.\ i = r\rho \end{cases}\end{array}$$

where $\cup$ distributes over $\vee$, i.e. $S \cup (S_1 \vee S_2) = (S_1 \vee S_2) \cup S = (S_1 \cup S) \vee (S_2 \cup S)$. Here Q is a support theory used only to ensure that the expansion of any infon results in a finite set; see theorem 4 below.

Theorem 4 *Let $P = P^1 \cup \{TD\}$, with P^1 being the type-1 theory in an $\mathbf{A}_1$ policy engine. Then, $\mathcal{E}_P(P, i)$ is a finite set for any infon i.*

Proof. Immediate, since the dependency graph of P^1 is acyclic, P^1 does not contain θ, and the Horn clause which encodes *TD* strictly decreases the number of θ functions. □

We write $\mathcal{E}(i)$ for $\mathcal{E}_P(P, i)$, when P is clear from the context.

Example 3.3 Consider the infon $i = can\_read(a, file)$ along with the type-1 theory of example 3.1. Then,

$$\mathcal{E}(i) = \{user(a), public(file)\} \vee \{admin(a), public(file)\} \vee \{admin(a), classified(file)\} \vee \{can\_read(a, file)\}$$

Write $\mathcal{E}(i) = g_1 \vee g_2 \vee g_3 \vee g_4$, with $g_1 = \{user(a), public(file)\}$, etc. The guard $\mathcal{E}(i)$ is interpreted as: $can\_read(a, file)$ holds, i.e. a can read *file* according to the RBAC system of example 3.1, iff either of the following conditions hold: $[g_1]$ $user(a)$ and $public(file)$ are known, or $[g_2]$ $admin(a)$ and $public(file)$ are known, or $[g_3]$ $admin(a)$ and $classified(file)$ are known, or $[g_4]$ $can\_read(a, file)$ is known via a policy inference outside the RBAC system. Similarly, $\mathcal{E}(can\_write(a, file)) = \{admin(a)\} \vee \{can\_write(a, file)\}$.

We refine the function ζ (introduced above) by incorporating the expansion function $\mathcal{E}$ into ζ. This intuitively ensures that $\zeta(i)$, for infon i, is obtainable from $\zeta(\sigma(a, i))$ if there exist at least one $g \in_\vee \mathcal{E}(\theta(a, i))$ such that $\zeta(g)$ can be obtained first. Hence, we define:

$$\zeta(i) = \begin{cases} \{\!|\zeta(X), sig(\bar{A}, \zeta(X))|\!\}_{\zeta(\mathcal{E}_P(P,\theta(A,X)))} & \text{if } i = \sigma(A, X) \\ \theta(\bar{A}, \zeta(X)) & \text{if } i = \theta(A, X) \\ i & \text{otherwise} \end{cases}$$

Here, $\{\!|x|\!\}_{k_1 \vee \cdots \vee k_\ell}$ stands for the tuple $\{\!|x|\!\}_{k_1}, \cdots, \{\!|x|\!\}_{k_\ell}$, function ζ distributes over $\vee$, and $P = P^1 \cup \{TD\}$ with P^1 being the type-1 theory at hand. For a finite set of infons g, $\zeta(g)$ is defined as the concatenation of $\zeta(i)$, for all $i \in g$. Note that elements of $\mathcal{E}_P(P, \theta(A, X))$ in the definition of ζ are singletons. This is because in any $\mathbf{A}_1$ policy engine, $\mathcal{E}_P(P, \theta(a, i)) = \mathcal{E}_{\{TD\}}(\{TD\}, \theta(a, i))$, as θ does not appear in P^1.

3.3.3 Correctness of the encoding

The following theorem ensures that if a fact is derivable in the logic program of an $\mathbf{A}_1$ policy engine, then its corresponding encoded term can be derived using the Dolev-Yao inference rules, and vice versa.

We consider the standard Dolev-Yao capabilities for the term algebra $\mathcal{T}_{\Sigma(\mathcal{V})}$, which comprises both infon and message constructors. That is, the infon constructors are seen as uninterpreted functions, while message constructors (e.g. $\{\!|\cdot|\!\}$.) have their standard meaning in the Dolev-Yao model. We write $T \vdash u$ for u is derivable from a set of terms T with the standard Dolev-Yao inference rules as formalized, e.g. [51].

Theorem 5 (Correctness) *Let P be the intensional knowledge of an* $\mathbf{A}_1$ *policy engine, with $P = \{TA\} \cup Q$, $Q = \{TD\} \cup P^1$ and P^1 being a type-1 theory. For any (ground) infon f and finite set of (ground) infons G:*

$$\mathcal{K}(f) \in \lceil \mathcal{K}(G) \rceil^P \iff \exists g \in_{\mathsf{v}} \mathcal{E}_Q(Q, f). \quad \zeta(G) \vdash \zeta(g)$$

where for $G = \{f_1, \cdots, f_\ell\}$, $\mathcal{K}(G)$ stands for $\{\mathcal{K}(f_1), \cdots, \mathcal{K}(f_\ell)\}$.

Proof. Fix the policy set P. We write $G \Vdash f$ for $\mathcal{K}(f) \in \lceil \mathcal{K}(G) \rceil^P$, suppress $\mathcal{K}$ when confusion is unlikely, and write $\mathcal{E}(f)$ for $\mathcal{E}_Q(Q, f)$. Below, we talk about *proof trees* for f, given G. The correspondence between finding proof trees and computing closures is immediate. The proof is split into two directions.

$\Rightarrow$ We use structural induction on proof trees for f, given G. If $f \in G$, then the implication is trivial. Otherwise, consider the last rule applied in the proof tree:

- (TA) Then $G \Vdash \sigma(a, f), \theta(a, f)$, for some $a \in \mathit{Agents}$. By induction hypotheses,

  $$\exists s \in_{\mathsf{v}} \mathcal{E}(\sigma(a, f)), t \in_{\mathsf{v}} \mathcal{E}(\theta(a, f)).\ \zeta(G) \vdash \zeta(s), \zeta(t).$$

  Observe that $s = \sigma(a, f)$. The term $\zeta(\sigma(a, f))$ is the tuple $\{\!|\, \zeta(f), \mathit{sig}(\overline{a}, \zeta(f)) \,|\!\}_{\mathcal{E}(\theta(a,f))}$. Since $t \in_{\mathsf{v}} \mathcal{E}(\theta(a, f))$, through unpairing, we obtain the cipher-text $\{\!|\, \zeta(f), \mathit{sig}(\overline{a}, \zeta(f)) \,|\!\}_{\zeta(t)}$ from $\zeta(\sigma(a, f))$. From $\zeta(G) \vdash \zeta(t)$, by applying the *Sdec* rule and unpairing we get $\zeta(G) \vdash \zeta(f)$. Clearly $f \in_{\mathsf{v}} \mathcal{E}(f)$.
- (TD) Then $f = \theta(a, \theta(b, i))$ for some $a, b \in \mathit{Agents}$ and $i \in \mathit{Infons}$, and $G \Vdash \theta(a, i)$. By induction hypotheses, $\exists t \in_{\mathsf{v}} \mathcal{E}(\theta(a, i))$. $\zeta(G) \vdash \zeta(t)$. Now, the claim follows since for any infon t, $t \in_{\mathsf{v}} \mathcal{E}(\theta(a, i))$ implies $t \in_{\mathsf{v}} \mathcal{E}(\theta(a, \theta(b, i)))$.
- (Type-1) Let $R = r \leftarrow r_1, \ldots, r_\ell \in P^1$ be the last rule applied. Then $f = r\rho$ and $G \Vdash r_1\rho, \cdots, r_\ell\rho$, for some grounding substitution ρ (cf. condition (b) in definition 11). By induction hypotheses, $\exists r'_1 \in_{\mathsf{v}} \mathcal{E}(r_1\rho), \cdots, r'_\ell \in_{\mathsf{v}} \mathcal{E}(r_\ell\rho)$. $\zeta(G) \vdash \zeta(r'_1), \cdots, \zeta(r'_\ell)$. By definition of $\mathcal{E}$, $\{r'_1, \cdots, r'_\ell\} \in_{\mathsf{v}} \mathcal{E}(f)$, hence follows the claim.

$\Leftarrow$ First, we claim that $\zeta(G) \vdash \zeta(g)$ implies $G \Vdash g$. Notice that the $\zeta(g)$ is either of the form $\{\!|\zeta(x), \mathit{sig}(\overline{a}, \zeta(x))|\!\}_{\mathcal{E}(\theta(a,x))}$, or of the form $i(x)$, with i being an infon constructor. The claim follows by case analysis on the Dolev-Yao attacker's message (de)composition abilities. In particular,

note that (1) to fabricate $\{\!|\zeta(x), sig(\overline{a}, \zeta(x))|\!\}_{\mathcal{E}(\theta(a,x))}$, the attacker needs to construct $sig(\overline{a}, \zeta(x))$, which is impossible as the attacker does not own the private key for any $\bar{a} \in \overline{Agents}$, and (2) infon constructors are uninterpreted functions in the Dolev-Yao model, i.e. they can neither be applied by the attacker, nor their application can be deconstructed. The other cases are straightforward; we thus omit them here. Finally, notice that if $G \Vdash g$ and $g \in_{\mathsf{v}} \mathcal{E}(f)$, then $G \Vdash f$; hence follows the claim.

This completes our proof. □

To conclude this section, note that the purpose of the proposed encoding is to replace the logic programs of services with the derivation rules of the Dolev-Yao model. Theorems 4 and 5 indicate that the computations in any $\mathbf{A}_1$ policy engine can be translated to finitely many proof searches in the Dolev-Yao attacker inference model, using the encoding introduced above.

As a result, intuitively, the attacker and all the services would be equipped with the reasoning power of the Dolev-Yao model, which is well understood and comes with decision algorithms for reachability. The encoding thus paves the way towards deciding reachability in security-sensitive services while accounting for both the communication and policy levels of services.

4 One-step Transition Decision Procedures

4.1 Logical background

We consider a finite set $\mathcal{F}$ of function symbols and a function $\alpha : \mathcal{F} \to \mathbb{N}$ called the *signature* function. Given a symbol $f \in \mathcal{F}$ the value $\alpha(f)$ is called the *arity* of f. Given a signature α, an α-algebra A is a pair (M_A, F_A) with a non-empty set M_A and a finite set of functions F_A such that for every function symbol f in the domain of α there exists in F_A a function $f_A : M_A^{\alpha(f)} \to M_A$.

Given a signature α of domain $\mathcal{F}$, a denumerable set of constants $\mathcal{C}$ and a denumerable set of variables $\mathcal{X}$, the free α-algebra generated by $\mathcal{X} \cup \mathcal{C}$ is called the *set of terms* over $\mathcal{F}$ and is denoted $\mathrm{T}_{\mathcal{C}}(\mathcal{F}, \mathcal{X})$. It is the least set T that contains $\mathcal{X}$, $\mathcal{C}$, and such that for every $f \in \mathcal{F}$ and every sequence $t_1, \ldots, t_{\alpha(f)}$ of elements of T we have $f(t_1, \ldots, t_{\alpha(f)}) \in T$.

As usual we consider terms built with function symbols (with arity) in a first-order signature $\mathcal{F}_t$ over a denumerable set of constants (denoted $\mathcal{C}$) and a denumerable set of variables (denoted $\mathcal{X}$). A term in which no variable occurs is *ground*. In the rest of this section any expression defined over terms (sets, states, clauses, etc.) in which only ground terms occur will also be called ground. Given a term $t \in \mathrm{T}_{\mathcal{C}}(\mathcal{F}, \mathcal{X})$ we denote $\mathrm{Var}(t)$ the set of variables (*i.e.* elements of $\mathcal{X}$) that occur in t. If $\mathrm{Var}(t) = \emptyset$ we say that the term t is *ground*. We denote $\mathrm{T}_{\mathcal{C}}(\mathcal{F})$ the set of ground terms.

We also assume that we are given a finite set $\mathcal{F}_p$ of relation symbols and an arity application $\beta : \mathcal{F}_p \to \mathbb{N}$. Each symbol $f_p \in \mathcal{F}_p$ is interpreted as a $\beta(f_p)$-ary relation on the set of terms $\mathrm{T}_{\mathcal{C}}(\mathcal{F}, \mathcal{X})$. The set of *facts* is the set:

$$\left\{ f_p(t_1, \ldots, t_{\beta(f_p)}) \,|\, t_1, \ldots, t_{\beta(f_p)} \in \mathrm{T}_{\mathcal{C}}(\mathcal{F}, \mathcal{X}) \right\}$$

A fact $f_p(t_1, \ldots, t_{\beta(f_p)})$ is *ground* if $t_1, \ldots, t_{\beta(f_p)} \in \mathrm{T}_{\mathcal{C}}(\mathcal{F})$. A *Possibly negated fact* is either a fact or the negation of a fact. A *clause* is a set of possibly negated facts. It is ground if it is a set of possibly negated ground facts. A clause is a *Horn clause* if it contains at most one fact. It is a *definite* Horn clause if it contains exactly one fact. Definite Horn clauses are usually written $f \leftarrow \Gamma$ where Γ is the set of facts whose negation occurs in the clause and f is the fact that occurs positively. If $\Gamma = \emptyset$ the clause is usually identified with its positive fact.

A *substitution* σ is an idempotent mapping from $\mathcal{X}$ to $\mathrm{T}_{\mathcal{C}}(\mathcal{F}, \mathcal{X})$ such that $\sigma(x) \neq x$ only for a finite subset of $\mathcal{X}$ called the *support* of σ. A substitution is ground if for every variable x in its support the term $\sigma(x)$ is ground. Substitutions are extended homomorphically to terms, facts, possibly negated facts and clauses. The application of a substitution will usually be denoted

in the postfix notation, *e.g.* $t\sigma$ is the term t in which every variable x occurring in the domain of σ is replaced by the term $\sigma(x)$. The result of the application of a substitution σ on a term, fact, possibly negated fact, or clause is called an *instance* of the latter.

A set of atoms $\mathcal{M}$ is a *model* of a set of ground possibly negated facts s if either a fact in s is also in $\mathcal{M}$ or a negated fact in s is not in $\mathcal{M}$. It is the model of a clause C if it is the model of every ground instance of C. It is the model of a set of clause $\mathcal{H}$ if it is a model of every clause in $\mathcal{H}$.

It is well-known that every set of Horn clauses is either inconsistent or has a least model for inclusion. In the latter case this model is also the least fixpoint obtained by computing the consequences of the subset of definite Horn clauses. It is unique but may not be finite.

4.2 Logical model of ASLan

4.2.1 States and transitions in ASLan

A *state* is a set of ground possibly negated facts. Unlike clauses, which represent disjunctions of facts, states represent conjunctions of facts. For the formal treatment we thus identify a state s with a set of Horn clauses $\mathcal{H}_s$ that each contains exactly one of the possibly negated facts in s.

Example 4.1 *For instance, we associate to the state* $s = \{q(x) \cdot \neg p(x)\}$ *the set of Horn clauses* $\mathcal{H}_s$ *defined as follows:*

$$\mathcal{H}_s = \begin{cases} q(s) \leftarrow \emptyset \\ \leftarrow p(x) \end{cases}$$

Given a state s and a set of Horn clauses $\mathcal{H}$ we denote $\lceil s \rceil^{\mathcal{H}}$ the least model for inclusion of the set of Horn clauses $\mathcal{H} \cup \mathcal{H}_s$. A *symbolic state* is a couple $(\mathcal{M}, C)$ where:

- $\mathcal{M}$ is a set of possibly negated facts;
- C is a conjunction of equalities and disequalities on terms.

A symbolic state is *simple* if the conjunction is over an empty set of equalities and disequalities. Given a symbolic state s we denote $\mathrm{Pos}(l)$ the set of facts occurring in s and $\mathrm{Neg}(s)$ the set of negated facts occurring in s.

A *transition rule* is a couple (l, r) denoted $l \Rightarrow r$ of symbolic states with:

$$\begin{cases} \mathrm{Var}(r) \subseteq \mathrm{Var}(\mathrm{Pos}(l)) \\ r = \mathrm{Pos}(r) \end{cases}$$

In accordance with D2.1 ([9], pp. 28–29) we say that a transition rule $l \Rightarrow r$ with $l = (\mathcal{M}, C)$ is *applicable* for a set of Horn clauses $\mathcal{H}$ on a state s with the ground substitution σ of support $\mathrm{Var}(\mathrm{Pos}(l))$ if for every ground substitution σ' of support $\mathrm{Var}(\mathrm{Neg}(l)) \setminus \mathrm{Var}(\mathrm{Pos}(l))$:

- for every fact $f \in \mathrm{Pos}(l)$ we have $f\sigma \in \lceil s \rceil^{\mathcal{H}}$ and for every negated fact $not(f)$ in $\mathrm{Neg}(l)$ we have $f\sigma\sigma' \notin \lceil s \rceil^{\mathcal{H}}$;
- $\sigma\sigma'$ satisfies all the equalities and disequalities in C.

The result of the application of a transition rule $l \Rightarrow r$ applicable on a state s is the state $s' = (s \setminus \mathrm{Pos}(l)\sigma) \cup r\sigma$. Given a set of transitions $\mathcal{P}$ we say that a state s is an *initial state* whenever there exists in $\mathcal{P}$ a transition rule $\emptyset \Rightarrow s$.

Example 4.2 *Consider the specification $\mathcal{S} = (\mathcal{H}, \mathcal{P})$ with:*

$$\begin{cases} \mathcal{H} &= \{p(x) \leftarrow q(x)\} \\ \mathcal{P} &= \{\emptyset \Rightarrow p(a).q(a)\ ,\ p(x).q(x) \Rightarrow q(x)\} \end{cases}$$

The state $s = p(a).q(a)$ is an initial state. Applying the second transition rule with the substitution $\{x \mapsto a\}$ yields a second state $s' = q(a)$. We note that we have $s' \neq s$ even though $\lceil s \rceil^{\mathcal{H}} = \lceil s' \rceil^{\mathcal{H}}$.

4.2.2 ASLan specifications

We can now define our model of ASLan specification.

Definition 12 *A* specification *is a couple $\mathcal{S} = (\mathcal{H}, \mathcal{P})$ where $\mathcal{H}$ is a finite set of Horn clauses and $\mathcal{P}$ is a finite set of transition rules.*

Given a specification $\mathcal{S} = (\mathcal{H}, \mathcal{P})$ a $\mathcal{S}$-execution is a sequence of states $s_0, \ldots, s_n$ such that:

- s_0 is an initial state;
- For $i \in \{1, \ldots, n\}$ there exists a transition rule in $\mathcal{P}$ from s_{i-1} to s_i.

We note that the restriction on the variables in transition rules implies that:

- An initial state is ground, *i.e.* contains no variables;
- Since the substitution σ with which the rule is applied must be ground, all states in a $\mathcal{S}$-trace are ground.

4.2.3 ASLan goals

The analysis of Web Services as conducted in the AVANTSSAR project consists in proving that a given specification has a given property, which is specified as a *goal* in ASLan. In the course of this project we have considered three types of properties:

Reachability properties: as specified in [9] these are goals that are expressed by a transition $s \Rightarrow \emptyset$ where s is a symbolic state representing the interesting states;

LTL properties: goals may also be expressed by LTL formulas that are evaluated on the possible traces of a specification. We leave the analysis of these properties to D3.2;

Structural properties: in particular when considering the service synthesis problem we have to consider goals that are evaluated against all the possible executions of an ASLan specification. Though the ASLan language does not currently provide a way to conveniently express these properties we have analyzed them in a variety of settings.

In order to simplify the problem we consider in the rest of this section only the reachability properties.

4.3 Relevant specifications

4.3.1 Web services and aspect-based programming

While Service Oriented Architectures aim at achieving reusability of software components and agility—*i.e.* an easy adaptation to changes in the execution environment or in the policies applicable to the service—of complex applications, other programming paradigms have been developed to ease the development of these software components.

Though the AVANTSSAR project is uniquely concerned with the validation of the service interface and not of the underlying application, one of these programming paradigm, aspect-based programming, is of particular importance to us. Indeed the WS-* stack (SOAP, WSDL, WS-Policy, etc.) is also based on aspects.

The core of the WS-* stack is the WSDL language that describes the network interface, *i.e.* its different operations, the messages it sends and receives, and the locations and protocols to employ to connect to it. This "message" level is intimately intertwinedwith a workflow aspect. For instance

the definition of the send and receive operations in BPEL rely on the definition of a WSDL file, and the corresponding WSDL files include an extension to support the notion of partners in BPEL.

As far as the AVANTSSAR project is concerned the main other aspect that pertains to the security analysis of services is the policy aspect. Policies can be attached at different points of a WSDL specification and alter the meaning of the WSDL part of the specification. Depending on this alteration we can distinguish between two types of policies. On the one hand, message-level policies change the format of the messages acceptable or sent by the service. On the other hand, access control policies do not change the actual format of the messages but instead specify conditions on the *payload* of messages abiding to this format under which a message can be accepted.

Our goal in this section is to formalize the distinction between these different aspects in order to, firstly, justify the restriction of some of the developed methods to only one of these aspects, and secondly, to be able to provide conditions under which the security analysis of a set of services with their policy can be conducted modularly in each of the different aspects.

Outline of the rest of this section. The setting we have adopted in ASLan intends to use the set of Horn clauses in a specification $\mathcal{S} = (\mathcal{H}, \mathcal{P})$ to specify the intruder deduction rules and the access control policy of the honest agents. But the same sets of clauses can also be employed to encode Turing machines, or to simulate any kind of computation. In order to restrict the analysis to the cases of matter to us we thus have to restrict the possible sets of Horn clauses that may appear in a specification.

We consider two kinds of restrictions. The first one aims at defining rigorously what an access control policy is. We propose one possible definition and prove that a consequence of it is that removing the access control policy does not reduce the possible executions of the services under scrutiny (Proposition 1). While it permits us to define the class of clauses for which we want to prove decision procedures this definition is not sufficient to prove that one can reason modularly on the access control aspect of the specification of Web Services. We thus propose a second restriction on admissible sets of Horn clauses, and call the abiding sets of clauses *well-formed*, that in contrast is purely technical and is based on prior works on modular reasoning for reachability problems.

4.3.2 Separation of the different aspects

There is a discrepancy between the goal of the AVANTSSAR project, which is to analyze the security of services on the one hand, and the expressiveness

of the ASLan language in which it is possible to express almost any kind of specification.

A first step towards providing decision procedures is thus to identify which subset of the possible, correct, ASLan specifications corresponds to the specification of a set of services. In doing so we will take a special care to delineate, in these specifications, the part that pertains to the access control policy of services, and how this access control aspect is woven into the rest of the specification of the services. Our main guide to characterize access control policies is that, as is mentioned above, that they restrict the possible executions of a service, but never modify parts of the service that were not introduced to model access control policies. To formalize this assumption we need to introduce a few definitions. First we partition the facts in states according to the aspect they are related to.

Existing Web Services standards separate the description of a service, of its control flow and state, and of its access control policy in different files *e.g.* written respectively in WSDL, BPEL4WS, and XACML. Accordingly we define three aspects, namely the *Web Service*, the *workflow* and the *access control* ones, that respectively represent the interaction between entities, their internal states and computations, and their access control policies.

Example 4.3 *In the standard translation from* ASLan++ *to* ASLan *the* wf-*facts are the* `state` *and* `contains` *fact that represent the states of the honest entities, the* ws-*facts are the* `iknows` *fact in the sense that they define the messages sent and received by he honest entities, and the access control policy facts would be among the additional facts. In the rest of this section we try to be as independent as possible from this standard translation, keeping only the* `iknows` *facts to denote the knowledge of the intruder.*

Definition 13 *We assume that the set of predicates $\mathcal{F}_p$ is partitioned into three sets $\mathcal{F}_{wf}$, $\mathcal{F}_{ws}$, and $\mathcal{F}_{ac}$ modelling respectively workflow related facts, Web Service related facts, and access control related facts.*

Accordingly we say that a fact $p(t_1, \ldots, t_n)$ is a *wf-fact* (resp. a *ws-fact*, a *ac-fact*) if $p \in \mathcal{F}_{wf}$ (resp. $p \in \mathcal{F}_{ws}$, $p \in \mathcal{F}_{ac}$).

It is now possible to specify rigorously how the access control aspect should be woven into the other parts of the application. Given a set of aspects $A \subseteq \{wf, ws, ac\}$ the function π_A, the *projection* to the A-facts, is the mapping from states to states such that:

$$\pi_A(s) = \{f \in s \mid f \text{ is an } a\text{-fact and } a \in A\}$$

We extend this projection to rules and specifications as follows. Given a specification $\mathcal{S} = (\mathcal{H}, \mathcal{P})$ and a set of aspects $A \subseteq \{wf, ws, ac\}$ let:

$$\begin{cases} \pi_A(\mathcal{H}) &= \{f \leftarrow \Gamma \in \mathcal{H} \mid f \text{ is a } a\text{-fact and } a \in A\} \\ \pi_A(\mathcal{P}) &= \{\pi_A(l) \Rightarrow \pi_A(r) \mid l \Rightarrow r \in \mathcal{P}\} \end{cases}$$

Given a specification $\mathcal{S}$ and a set of aspects A we define the A-version of $\mathcal{S}$ to be the specification $\pi_A(\mathcal{S}) = (\pi_A(\mathcal{H}), \pi_A(\mathcal{P}))$.

4.3.3 Web Service specifications (WS specifications)

We are now equipped to define precisely the class of ASLan specification that pertains to the modelling of Web Services.

Definition 14 *(WS-specifications) We say that a specification $(\mathcal{H}, \mathcal{P})$ is a* Web Service specification *whenever:*

wf-facts restrictions: *We have*

- (wf_H) $\pi_{wf}(\mathcal{H}) = \emptyset$,
- (wf_T) *for every transition rule* $l \Rightarrow r \in \mathcal{P}$ *we have* $Var(\pi_{wf}(r)) \subseteq Var(\mathrm{Pos}(\pi_{ws}(l))) \cup Var(\mathrm{Pos}(\pi_{wf}(l)))$;

ac-facts restrictions:

- (ac_H) *For each clause* $f \leftarrow \Gamma \in \mathcal{H}$ *if* f *is a ws-fact then no fact in* Γ *is an ac-fact*
- (ac_T) *For each transition rule* $l \Rightarrow r \in \mathcal{P}$ *either* $Var(\pi_{ac}(l)) \subseteq Var(\mathrm{Pos}(\pi_{ws}(l))) \cup Var(\mathrm{Pos}(\pi_{wf}(l)))$ *or* $\pi_{\{wf,ws\}}(l) = \pi_{\{wf,ws\}}(r) = \emptyset$.

This definition is the formal translation of the following hypotheses:

(wf_H): The constraint $\pi_{wf}(\mathcal{H}) = \emptyset$ implies that the internal computations of the services are not encoded in Horn clauses;

(wf_T): With the exception of nonce creation, for which we assume the variables quantified existentially are skolemized when a transition rule is applied, a stateful service should only operate on data it has initially access to or on data received. In ASLan specifications this leads to the restriction $\mathrm{Var}(\mathrm{Pos}(\pi_{wf}(r))) \subseteq \mathrm{Var}(\mathrm{Pos}(\pi_{ws}(l))) \cup \mathrm{Var}(\mathrm{Pos}(\pi_{wf}(l)))$. Another way to express the restriction is to say that once all ac-facts are removed from transition rules we still have regular transition rules;

(ac_H): The access control rules may limit the possible execution of the services by changing the truth value of ac-facts, but do not change the truth value of ws- or wf-facts. This property is stated formally in Proposition 1;

(ac_T): We have to model that an entity queries the access control system by forming a request from the data it has access to and that the access control is stateful and may evolve *independently* from the applications, as is the case *e.g.* in delegation and revocation. The first condition permits one to express that the entity takes the access control decision into account, while the second condition implies that only access control related objects are affected by a transition, thereby modelling an evolution of the access control system.

4.4 Reachability problems

4.4.1 Definition

In order to simplify the presentation we consider only ASLan goals that are defined by attack states as described in Deliverable 2.1 ([9], p. 29). An attack state s is represented in ASLan with a transition rule $s \Rightarrow s.\texttt{attack}$. The security property is violated whenever there exists an execution of the services in which this transition rule is fired.

As a result, the security analysis of a specification amounts to deciding whether the transition rules in a specification—including the transition rules encoding security property violations—can be applied in a given order.

Reachability$_{\mathcal{H}}(\mathcal{P}, (l_i \Rightarrow r_i)_{1\leq i\leq n})$

Input: A specification $\mathcal{S} = (\mathcal{H}, \mathcal{P})$ and a finite sequence $(l_i \Rightarrow r_i)_{1\leq i\leq n}$ of rules of $\mathcal{P}$.

Output: SAT if there exists a substitution σ and a $\mathcal{S}$-trace $(s_i)_{0\leq i\leq n}$ with $s_0 = \emptyset$ such that for $1 \leq i \leq n$ the instance $l_i\sigma \Rightarrow r_i\sigma$ is a transition from s_{i-1} to s_i.

The AVANTSSAR tools can be constrained to solve a typed version of this problem in which each variable can only be instantiated by a finite number of ground terms. Tools such as SATMC solve this adapted problem by considering all the possible ground instances of the transition rules. This eager instantiation reduces the problem to one in which ground instances of the transition rules are given.

GroundReachability$_{\mathcal{H}}(\mathcal{P}, (l_i \Rightarrow r_i)_{1\leq i\leq n})$

Input: A specification $\mathcal{S} = (\mathcal{H}, \mathcal{P})$ and a sequence $(l_i \Rightarrow r_i)_{1 \leq i \leq n}$ of transition rules that are ground instances of rules in $\mathcal{P}$.

Output: SAT if there exists a $\mathcal{S}$-trace $(s_i)_{0 \leq i \leq n}$ with $s_0 = \emptyset$ such that for $1 \leq i \leq n$ the transition rule $l_i \Rightarrow r_i$ is a transition from s_{i-1} to s_i.

We define in the rest of this section properties of the set of Horn clauses $\mathcal{H}$ such that the reachability and ground reachability problems for specifications $(\mathcal{H}, \mathcal{P})$ are decidable.

4.4.2 Reachability problems for WS-specifications

In Subsection 4.3.3 we have defined Web Service specifications. At that point we have informally justified that the given definitions did reflect the properties of access control policies *w.r.t.* the other aspects. Next proposition gives a formal meaning to that assertion. Though we may have over-constrained the Web Service specifications we believe this setting satisfactorily captures how the different aspects are woven into Web Services.

Proposition 1 *Let $\mathcal{S} = (\mathcal{H}, \mathcal{P})$ be a Web Service specification. If $s_0, \ldots, s_n$ is a $\mathcal{S}$-trace then $\pi_{\{wf,ws\}}(s_0), \ldots, \pi_{\{wf,ws\}}(s_n)$ is a $\pi_{\{wf,ws\}}(\mathcal{S})$-trace.*

PROOF. *By induction on n. The base case $n = 0$ is trivial. Assume the proposition holds for any $\mathcal{S}$-trace of length bounded by n and consider a $\mathcal{S}$-trace $s_0, \ldots, s_{n+1}$. By induction we already know that $\pi_{\{wf,ws\}}(s_0), \ldots, \pi_{\{wf,ws\}}(s_n)$ is a $\pi_{\{wf,ws\}}(\mathcal{S})$-trace. If the transition rule $l \Rightarrow r$ applied on s_n to obtain s_{n+1} is such that $\pi_{\{wf,ws\}}(l) = \emptyset$ then by the point (ac_T) of Definition 14 we also have $\pi_{\{wf,ws\}}(r) = \emptyset$, and thus $\pi_{\{wf,ws\}}(s_{n+1}) = \pi_{\{wf,ws\}}(s_n)$ by definition of the transition relation. Thus one can extend the trace with a stutter.*

Thus to prove the theorem it suffices to prove that a wf- or ws-fact f is in $\lceil s_n \rceil^{\mathcal{H}}$ if, and only if, it is in $\left\lceil \pi_{\{wf,ws\}}(s_n) \right\rceil^{\pi_{\{wf,ws\}}(\mathcal{H})}$, i.e.:

$$\pi_{\{wf,ws\}}(\lceil s_n \rceil^{\mathcal{H}}) = \pi_{\{wf,ws\}}(\left\lceil \pi_{\{wf,ws\}}(s_n) \right\rceil^{\pi_{\{wf,ws\}}(\mathcal{H})})$$

By definition of Web Service specifications, point (ac_H), Horn clauses whose head is a ws-fact have only wf- or ws-facts in their body, and by point (wf_H) there are no Horn clauses whose head is a wf-fact. We thus have:

$$\pi_{\{wf,ws\}}(\mathcal{H}) \subseteq \mathcal{H}$$

Together with the trivial inclusion $\pi_{\{wf,ws\}}(s_n) \subseteq s_n$ *this immediately yields the inclusion* $\left\lceil \pi_{\{wf,ws\}}(s_n) \right\rceil^{\pi_{\{wf,ws\}}(\mathcal{H})} \subseteq \pi_{\{wf,ws\}}(\lceil s_n \rceil^{\mathcal{H}})$.

To prove the converse inclusion we consider a well-founded total ordering $<$ *on the facts in* $\lceil s_n \rceil^{\mathcal{H}}$ *such that for each fact* f *in this set either* f *is minimal for the ordering and in* s *or there exists facts* $f_1, \ldots, f_k$ *such that:*

$$\begin{cases} f_i < f & \text{for } 1 \leq i \leq k \\ \{f_1, \ldots, f_k\} \rightarrow f & \text{is an instance of a clause in } \mathcal{H} \end{cases}$$

This ordering is well defined given our closed world assumption, and is well-founded since the closure is defined as a least fix point. By contradiction if $\pi_{\{wf,ws\}}(\lceil s_n \rceil^{\mathcal{H}}) \not\subseteq \left\lceil \pi_{\{wf,ws\}}(s_n) \right\rceil^{\pi_{\{wf,ws\}}(\mathcal{H})}$ *the set* $\pi_{\{wf,ws\}}(\overline{s_n}^{\mathcal{H}}) \setminus \overline{\pi_{\{wf,ws\}}(s_n)}^{\pi_{\{wf,ws\}}(\mathcal{H})}$ *is not empty. Thus it has a minimal element* f *for the well-founded ordering* $<$*. If this element is a minimal element for the ordering, i.e.* $f \in s_n$*, we also have* $f \in \pi_{\{wf,ws\}}(s_n)$*, a contradiction. Thus* f *is not minimal, and thus cannot be a wf-fact.*

Thus f *must be a ws-fact which is not minimal in* $\lceil s_n \rceil^{\mathcal{H}}$*. Thus by definition of the ordering* $<$ *there exists facts* $f_1, \ldots, f_k$ *such that* $\{f_1, \ldots, f_k\} \rightarrow f$ *is the instance of a Horn clause in* $\mathcal{H}$*. Since this is a Web Service specification and the head of this clause is not an ac-fact it is also in* $\pi_{\{wf,ws\}}(\mathcal{H})$*. Since* $f \notin \left\lceil \pi_{\{wf,ws\}}(s_n) \right\rceil^{\pi_{\{wf,ws\}}(\mathcal{H})}$ *at least one of these facts—w.l.o.g. let us assume this is* f_1*—is not in* $\left\lceil \pi_{\{wf,ws\}}(s_n) \right\rceil^{\pi_{\{wf,ws\}}(\mathcal{H})}$*. Since it is a wf- or ws-fact we note that* f_1 *is in* $\pi_{\{wf,ws\}}(\lceil s_n \rceil^{\mathcal{H}})$*. This contradicts the minimality of* f*, and thus the non-emptiness of* $\pi_{\{wf,ws\}}(\lceil s_n \rceil^{\mathcal{H}}) \setminus \left\lceil \pi_{\{wf,ws\}}(s_n) \right\rceil^{\pi_{\{wf,ws\}}(\mathcal{H})}$.

4.4.3 The case of ground reachability problems

Before proceeding to the description of more complex procedures let us first consider the case of ground reachability problems. We recall that solving this problem is sufficient for the typed analysis of Web Services. Let $\mathcal{S} = (\mathcal{H}, \mathcal{P})$ be a specification such that every rule in $\mathcal{P}$ is ground.

Theorem 6 *(Reduction of ground reachability to ground entailment) If the ground entailment problem for the theory* $\mathcal{H}$ *is decidable then* $\mathbf{GroundReachability}_{\mathcal{H}}(\mathcal{P}, (l_i \Rightarrow r_i)_{1 \leq i \leq n})$ *is decidable.*

Proof sketch. *The assumption that the ground entailment problem for* $\mathcal{H}$ *is decidable means exactly that one can decide whether a ground clause* $\Gamma \rightarrow \Delta$ *is a consequence of* $\mathcal{H}$*. Noting that the ground rule* $l_i \Rightarrow r_i$ *can be*

applied on the ground state s if, and only if, every ground fact f in $l_i \setminus s$ is a logical consequence of the set of facts F in s and of the theory $\mathcal{H}$. Deciding whether this is the case is the same as deciding whether $\mathcal{H}$ entails the ground clause $F \rightarrow f$.

We give in [30] a criterion on $\mathcal{H}$ ensuring the decidability of the ground entailment problems. Let us note however that other criteria can be found in the literature [10].

5 Process equivalence

5.1 Introduction

Context. Well-known stories about flaw discoveries have revealed that protocols and services may be subject to unexpected and undesirable behaviours under malevolent attacker actions. Formal analysis is therefore mandatory for gaining the level of confidence required in critical applications. Formal methods and related tools have proved to be successful to some extent for this task. But they are limited in expressiveness since in most cases the focus has been on the resolution of reachability problems, and as a consequence very few effective procedures consider the more general case of equivalence properties useful for modelling many important security properties.

For instance, the fundamental compositionality results on cryptographic protocols by Canetti et al. (e.g., [28, 48]) require to prove that a process is equivalent to an ideal abstraction of it in every environment. Hence deciding equivalence of processes is mandatory to obtain automated compositionality proofs.

We give in this section an alternative, and we believe simpler, proof of a deep result by Mathieu Baudet [13], namely that the equivalence of symbolic constraints is decidable for deduction systems on a finite signature modulo a subterm convergent equational theory.

Motivation. Observational equivalence is a crucial notion for specifying security properties such as anonymity or secrecy of a ballot in vote protocols [36]. For instance, observational equivalence can justify that there is no action of an attacker that makes distinguishable two protocol executions with different identities or vote values.

To be of effective use the notion of observational equivalence should be considered on processes modeling cryptographic protocols. We consider in this section finite processes called symbolic derivations that represent any (combination of) honest agents. One easily sees that these symbolic derivations encode plain processes of the applied π-calculus without replication nor branching ("else branches"). Since we consider only sequential attacker derivations, our notion of symbolic equivalence corresponds to trace equivalence of simple processes without branch nor replication of [35]. In the latter paper the authors proved that trace and observational equivalence coincide on this class of processes.

The only decidability result on the equivalence of symbolic traces we are aware of is the *coNP* decision procedure for S-equivalence for the class of subterm deduction systems and was given by Baudet [12, 13]. We propose

an alternative decision procedure for the same class of protocols that can be handled by Baudet. The complexity is the same for the subclass COMPILED_EQ of protocols that can be obtained by compiling Alice-and-Bob specifications. A more efficient procedure is presented in [29] when one considers only the Dolev-Yao deduction system. In spite of the relevance of this problem for the analysis of e.g., voting protocols, we are not aware of any extension of Baudet's results to other classes of deduction systems.

Applications. The equivalence notion we consider in this section has two straightforward applications, one related to the symbolic validation of cryptographic properties and one related to the search for on-line guessing attacks.

An *on-line* attack is one in which the attacker interacts with honest agents to achieve his goals which usually are the acquisition of a previously unknown piece of data, or the impersonation of a honest agent. In these cases the achievability of a goal can be reduced to a reachability problem. However, one may consider goals for which this reduction does not hold. For example, the dictionary attacks introduced by Schneier [57] consist in guessing a piece of data (usually a password) and interacting with the honest agents using this piece of data. Depending on the resulting communication the attacker knows whether the guess was correct. It is often the case that such attacks can be detected by the honest agents involved. For example, sending a wrong password will be detected by an authentication system that, after a small number of failures, may invalidate the account and ask for a new password. To take into account this possible response by honest agents, Ding and Horster [38] have introduced the concept of *undetectable on-line password guessing attacks*: *An attacker ... verifies the correctness of his guess using responses of S. ... A failed guess can not be detected and logged by S, as S is not able to depart an honest request from a malicious request.* Guessing attacks have been formalized thanks to the concept of indistinguishability (see e.g., [1]). We can say now that a protocol is vulnerable to undetectable on-line guessing attacks whenever *(i)* the honest agents cannot distinguish between a session with the right piece of data and one involving a wrong guess, whereas *(ii)* the intruder can distinguish the two executions. We model the first point by stating that the tests performed by the honest agents succeed in both cases, and the second point by saying that the two executions are not equivalent.

Inspired by the pioneering paper by Abadi and Rogaway in 2000 [4] several recent works have shown that computational proofs of indistinguishability ensuring the security of a protocol can be derived, under some natural hypothesis on cryptographic primitives, from symbolic proofs. This has opened the path to the automation of computational proofs. It was shown

by [34] that *in presence of an active attacker* observational equivalence of the symbolic processes can be transferred to the computational level. Hence symbolic equivalence seems to be the proper setting for deriving soundness results.

Related work Many works have been dedicated to proving correctness properties of cryptographic protocols using equivalences on process calculi. In particular, *framed bisimilarity* has been introduced by Abadi and Gordon [3] for this purpose, for the spi-calculus. Another approach that circumvents the context quantification problem is presented in [24] where labelled transition systems are constrained by the knowledge the environment has of names and keys. This approach allows for more direct proofs of equivalence.

To the best of our knowledge, the only tool capable of verifying equivalence-based secrecy is the resolution-based algorithm of ProVerif [18] that has been extended for this purpose. Proverif has also been enhanced for handling equivalences of processes that differ only in the choice of some terms in the context of the applied π-calculus [20]. This allows to add some equational theories for modelling properties of the underlying cryptographic primitives.

Few decidability results are available. In the article [47] Hüttel proves decidability for a fragment of the spi-calculus without recursion for framed bisimilarity. Since [35] relies on the proof of Baudet's result, which is long and difficult [13], we believe that a simpler way to get decidability of S-equivalence, as in our approach, may help to obtain new decidable cases for S-equivalence as well as for observational equivalence by the transfer results of [35].

Organization of this section. In Section 5.2 we recall basic notions on terms, equational theories, and subterm deduction systems which are used for modelling both honest and dishonest agents of cryptographic protocols. In Section 5.3 we introduce the central concept of symbolic derivation which can be considered as a specification for an agent behaviour. The solutions of a symbolic derivation are defined. They characterize the possible interactions of an agent with its environment. We show how we can express the more classical notion of static equivalence within the framework of symbolic derivations. The main result of the section is proved in Section 5.4, namely that *equivalence of symbolic derivations is decidable for subterm deduction systems.*

5.2 Definitions

5.2.1 Terms

We assume given a signature $\mathcal{F}$, an infinite set of variables $\mathcal{X}$ and two infinite sets of free constants C and $\mathrm{C}_{\mathrm{new}}$. The first set is intended to represent "regular" constants employed by the honest users while the second set $\mathrm{C}_{\mathrm{new}}$ is intended to represent fresh values created by an attacker. The set of terms built with $\mathcal{F}$, the constants and $\mathcal{X}$ is denoted $\mathrm{T}(\mathcal{F},\mathcal{X})$ and its subset of ground terms (terms without variables) $\mathrm{T}(\mathcal{F})$. We denote $\mathrm{Var}(t)$ (resp. $\mathrm{Const}(t)$) the set of variables (resp. constants) occurring in a term $t \in \mathrm{T}(\mathcal{F},\mathcal{X})$ and $\mathrm{Sub}(t)$ the set of subterms of t. These notations are extended as expected to sets of terms. We denote $t[s]$ a term t that admits s as subterm. Positions in a term t are defined as usual as finite sequences of positive integers. The empty sequence is denoted ε. We denote the subterm of t at position p with $t_{|p}$ which is defined by $t_{|\varepsilon} = t$ and if $t = f(t_1,\ldots,t_n)$ we define $t_{|i\cdot p} = (t_i)_{|p}$. The expression $t[p \leftarrow s]$ denotes a term obtained from term t by replacing the subterm at position p by s. The size $\|t\|$ of a term t is the number of distinct subterms of t. The notation is extended as expected to sets of terms.

A substitution σ is an idempotent mapping from $\mathcal{X}$ to $\mathrm{T}(\mathcal{F},\mathcal{X})$ such that $\mathrm{Supp}(\sigma) = \{x \,|\, \sigma(x) \neq x\}$, the *support* of σ, is a finite set. The application of a substitution σ to a term t (resp. a set of terms E) is denoted $t\sigma$ (resp. $E\sigma$) and results in the term t in which all variables x are replaced by the term $x\sigma$. A substitution σ is *ground* w.r.t. $\mathcal{F}$ if the image of $\mathrm{Supp}(\sigma)$ is included in $\mathrm{T}(\mathcal{F})$.

A rewriting system $\mathcal{R}$ is a finite set of couples $(l,r) \in \mathrm{T}(\mathcal{F},\mathcal{X}) \times \mathrm{T}(\mathcal{F},\mathcal{X})$, where each couple is called a *rewriting rule* and is denoted $l \to r$. The rewriting relation $\to_{\mathcal{R}}$ between terms is defined by $t \to_{\mathcal{R}} t'$ if there exists $l \to r \in \mathcal{R}$, a position p, and a substitution σ such that $l\sigma = t_{|p}$ and $r\sigma = s$, and $t' = t[p \leftarrow s]$. A rewriting system is *terminating* if for any term t there is no infinite sequence of rewritings starting from t. It is *convergent* if it has in addition the *confluence* property: every sequence of rewriting starting from t ends in the same term denoted $(t)\!\downarrow_{\mathcal{R}}$, or simply $(t)\!\downarrow$ if $\mathcal{R}$ is clear from the context. We say that a term t is in *normal form* if $t = (t)\!\downarrow_{\mathcal{R}}$. A substitution σ is in normal form if for all $x \in \mathrm{Supp}(\sigma)$, the term $\sigma(x)$ is in normal form. Given a substitution σ, we denote $(\sigma)\!\downarrow_{\mathcal{R}}$ the substitution such that, for all $x \in \mathrm{Supp}(\sigma)$ we have $(x\sigma)\!\downarrow_{\mathcal{R}} = x(\sigma)\!\downarrow_{\mathcal{R}}$.

A convergent rewriting system $\mathcal{R}$ defines an *equational theory* $\mathcal{E}$ which is a congruence relation on terms in $\mathrm{T}(\mathcal{F},\mathcal{X})$. We denote $t =_{\mathcal{E}} t'$ the fact that $(t)\!\downarrow = (t')\!\downarrow$ and say in this case that t and t' are equal modulo $\mathcal{E}$. We

say that the equational theory defined by $\mathcal{R}$ is *subterm* if $\mathcal{R}$ is furthermore such that for every rewrite rule $l \to r \in \mathcal{R}$ one has $r \in \mathrm{Sub}(l)$. We note that convergent subterm equational theories are always consistent, *i.e.*, they always have a model with more than one element.

Let $\mathcal{E}$ be a set of axioms on $\mathrm{T}(\mathcal{F}, \mathcal{X})$. An *$\mathcal{E}$-Unification system* $\mathcal{S}$ is a finite set of equations in $\mathrm{T}(\mathcal{F}, \mathcal{X})$ denoted by $(t_i \stackrel{?}{=} u_i)_{i\in\{1,\ldots,n\}}$. It is satisfied by a ground substitution σ, and we write $\sigma \models_{\mathcal{E}} \mathcal{S}$, if for all $i \in \{1, \ldots, n\}$ the equality $t_i\sigma =_{\mathcal{E}} u_i\sigma$ holds. When the equational theory $\mathcal{E}$ can be defined by a rewriting system that contains no equations we say that $\mathcal{E}$ is the empty theory, A unification system in the empty theory is also called a *syntactic unification system*. An important property of unification systems, whose proof can be found in [31], is the following replacement property.

Given terms u, v, t, we denote by $t\delta_{u,v}$ the parallel replacement of all occurrences of u by v in t. Given a substitution σ we denote by $\sigma\delta_{u,v}$ the substitution such that $x(\sigma\delta_{u,v}) = \sigma(x)\delta_{u,v}$.

Lemma 4 *For any equational theory $\mathcal{E}$, if an $\mathcal{E}$-unification system $\mathcal{S}$ is satisfied by a substitution σ, and c is any free constant not in $\mathcal{S}$, then for any term t, $\sigma\delta_{c,t}$ is also a solution of $\mathcal{S}$.*

For the empty theory this lemma admits a kind of reciprocal:

Lemma 5 *If σ satisfies an $\emptyset$-unification system $\mathcal{S}$ and for all $s \in \mathrm{Sub}(\mathcal{S})$ we have $s\sigma \neq t$ then for any constant c not occurring in t, $(s\sigma)\delta_{t,c} = s(\sigma\delta_{t,c})$. Hence $\sigma\delta_{t,c}$ is also a solution of $\mathcal{S}$.*

Proof. By structural induction on term s. If s is a constant, $s\sigma \neq t$ implies $s \neq t$ and thus $s = (s\sigma)\delta_{t,c} = s(\sigma\delta_{t,c})$. If s is a variable we simply apply the definition of replacement to get $(s\sigma)\delta_{t,c} = s(\sigma\delta_{t,c})$. If $s = f(s_1, \ldots, s_n)$, $s\sigma \neq t$ implies $(f(s_1, \ldots, s_n)\sigma)\delta_{t,c} = f((s_1\sigma)\delta_{t,c}, \ldots, (s_n\sigma)\delta_{t,c})$ and we apply the induction hypothesis to $(s_i\sigma)\delta_{t,c}$. □

5.2.2 Subterm Deduction Systems

We give a formal model for roles and the execution of roles (including the intruder). Messages are ground terms and deduction rules are rewrite rules on sets of messages representing the knowledge of an agent. Each role derives new messages from a given (finite) set of messages by using deduction rules. Furthermore, these deductions are considered *modulo* the equational congruence $=_{\mathcal{E}}$ generated by a set $\mathcal{E}$ of equational axioms satisfied by the function symbols of the signature $\mathcal{F}$.

Definition 1 *A* subterm deduction system *is a triple* $\langle \mathcal{F}, \mathcal{P}, \mathcal{E} \rangle$ *such that:*

- $\mathcal{F}$ *is a signature admitting a subset* $\mathcal{F}'$ *of* public symbols*;*
- $\mathcal{P} \subseteq \mathcal{F}$ *is a set of* public symbols*;*
- $\mathcal{E}$ *is defined by a* convergent *rewrite system* $\mathcal{R}$

such that each public symbol f *defines a deduction rule* $x_1, \ldots, x_n \twoheadrightarrow f(x_1, \ldots, x_n)$ *whose intended meaning is that given terms* $t_1, \ldots, t_n$ *it is possible to apply the function* f *on these terms to deduce a new term* $(f(t_1, \ldots, t_n))\!\downarrow$.

Example 5.1 *Let us now present the example of the Dolev-Yao deduction system with pairs and asymmetric encryption. The signature* $\mathcal{F}_{\mathcal{D}}$ *contains 3 public symbols of arity 2, namely* $\langle \_, \_ \rangle$, $\mathrm{ae}(\_, \_)$, *and* $\mathrm{ad}(\_, \_)$ *describing respectively the concatenation of two messages, the encryption of a message (its first argument) by a public key encryption algorithm where the key is the second message and the converse operation of decryption with the private key. It also contains two public projection symbols of arity 1, namely* $\pi_1(\_), \pi_2(\_)$, *and two symbols of arity 1, namely* $\mathrm{pk}(\_)$ *and* $\mathrm{sk}(\_)$ *denoting respectively public and private keys.*

All these symbols can be employed by any agent, and we have thus the following deduction rules:

$$\mathcal{P}_{\mathcal{D}} = \left\{ \begin{array}{lll|lll} \multicolumn{3}{c|}{\textit{Concatenation}} & \multicolumn{3}{c}{\textit{Encryption}} \\ x, y & \twoheadrightarrow & \langle x, y \rangle & x, y & \twoheadrightarrow & \mathrm{ae}(x, y) \\ x & \twoheadrightarrow & \pi_1(x) & x, y & \twoheadrightarrow & \mathrm{ad}(x, y) \\ x & \twoheadrightarrow & \pi_2(x) & & & \end{array} \right.$$

The equational theory $\mathcal{H}_{\mathcal{D}}$ *contains the following relations:*

$$\mathcal{E}_{\mathcal{D}} = \left\{ \begin{array}{lll|lll} \multicolumn{3}{c|}{\textit{Concatenation}} & \multicolumn{3}{c}{\textit{Encryption}} \\ \pi_1(\langle x, y \rangle) & = & x & \mathrm{ad}(\mathrm{ae}(x, \mathrm{pk}(y)), \mathrm{sk}(y)) & = & x \\ \pi_2(\langle x, y \rangle) & = & y & & & \end{array} \right.$$

The deduction system $\mathcal{I}_{\mathcal{DY}} = \langle \mathcal{F}_{\mathcal{D}}, \mathcal{P}_{\mathcal{D}}, \mathcal{E}_{\mathcal{D}} \rangle$ *describes the classical Dolev-Yao equational model with pairing and asymmetric encryption.*

Let us recall an interesting property of any subterm convergent theory $\mathcal{E}$. Let us consider a term t and a substitution σ in normal form. Since the equational theory is convergent, a bottom-up sequence of rewriting leads from $t\sigma$ to $(t\sigma)\!\downarrow$. We note that since this sequence is bottom-up and since σ is in normal form:

- Once a rewrite rule is applied at a position p, and since r is a strict subterm of l, all subterms below or at the position p in the resulting term are in normal form;
- As a consequence, the rewrite rules in the bottom-up sequence of rewritings on $t\sigma$ are applied at most once at each position p in t.

This second point implies that basic narrowing [46] terminates and, since $\mathcal{E}$ is convergent, is a complete unification procedure for $\mathcal{E}$. In addition we have that the number of narrowing steps to apply on a term t is smaller than $|\mathrm{Sub}(t) \setminus \mathrm{Var}(t)|$.

5.3 Symbolic Derivations

5.3.1 Definitions

Symbolic derivations. Given a deduction system $\langle \mathcal{F}, \mathcal{P}, \mathcal{E} \rangle$, a role applies public symbols in $\mathcal{P}$ to construct a response from its initial knowledge and from messages received so far. Additionally, it may test equalities between messages to check the well-formedness of a message. Hence the activity of a role can be expressed by a fixed symbolic derivation:

Definition 2 *(Symbolic Derivations) A symbolic derivation for a deduction system $\langle \mathcal{F}, \mathcal{P}, \mathcal{E} \rangle$ is a tuple $(\mathcal{V}, \mathcal{S}, \mathcal{K}, \mathrm{In}, \mathrm{Out})$ where $\mathcal{V}$ is a mapping from a finite ordered set $(\mathrm{Ind}, <)$ to a set of variables $\mathrm{Var}(\mathcal{V})$, $\mathcal{K}$ is a set of ground terms (the initial knowledge) In is a subset of Ind, Out is a multiset of elements of Ind and $\mathcal{S}$ is a set of equations.*

The set Ind represents internal states of the symbolic derivation. We impose that any $i \in \mathrm{Ind}$ denotes a state of one of the following kinds:

Deduction state: *There exists a public symbol $f \in \mathcal{P}$ of arity n such that $\mathcal{S}$ contains the equations $\mathcal{V}(i) \stackrel{?}{=} f(\mathcal{V}(\alpha_1), \ldots, \mathcal{V}(\alpha_n))$ with $\alpha_j < i$ for $j \in \{1, \ldots, n\}$.*

Re-use state: *Otherwise, if there exists $j < i$ with $\mathcal{V}(j) \stackrel{?}{=} \mathcal{V}(i)$;*

Memory state: *Otherwise, if there exists t in $\mathcal{K}$ and an equation $\mathcal{V}(i) \stackrel{?}{=} t$ in $\mathcal{S}$;*

Reception state: *Otherwise, we must have $i \in \mathrm{In}$;*

Additionally, a state i is also an emission state *if $i \in \mathrm{Out}$.*

A symbolic derivation is closed *if it has no reception state. A substitution σ* satisfies *a closed symbolic derivation if $\sigma \models_{\mathcal{E}} \mathcal{S}$.*

Remark on re-use states. We believe that using symbolic derivations instead of more standard constraint systems permits one to simplify the proofs by having a more homogeneous framework. There is however one drawback to their usage. While most of the time it is convenient to have an identification between the order of deduction of messages and their send/receive order, building in this identification too strictly would prevent us from expressing simple problems. Re-use states are employed to reorder the deduced messages to fit an order of sending messages which can be different. For example consider an intruder that knows (after reception) two messages a and b received in that order, and that he has to send first b, then a. Since the states in a symbolic derivation have to be ordered, we have to use at least one re-use state (for a) to be able to consider a sending of a *after* the sending of b. We note that re-use states that are not employed in a connection can be safely eliminated without changing the deductions, the definition of the knowledge nor the tests in the unification system.

Example 5.2 *Let us consider the cryptographic protocol for deduction system $\mathcal{I}_{\mathcal{DY}}$ where $\mathcal{F}_{\mathcal{D}}$ and $\mathcal{P}_{\mathcal{D}}$ have been extended by a free public symbol f:*

$A \to B$: $\mathrm{ae}(N_a, \mathrm{pk}(B))$
$B \to A$: $\mathrm{ae}(f(N_a), \mathrm{pk}(A))$
where
A ***knows*** $A, B, \mathrm{pk}(B), \mathrm{pk}(A), \mathrm{sk}(A)$
B ***knows*** $A, B, \mathrm{pk}(A), \mathrm{pk}(B), \mathrm{sk}(B)$

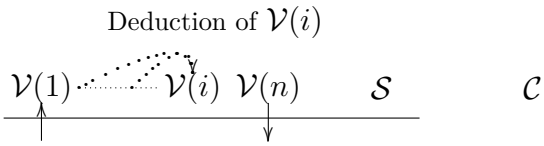
Let us define a symbolic derivation for role B:

$$\begin{array}{rcl} \textsc{Ind} & = & \{0, \ldots, 8\} \\ \mathcal{V} & = & i \in \textsc{Ind} \mapsto x_i \\ \mathcal{K} & = & \{A, B, \mathrm{pk}(A), \mathrm{pk}(B), \mathrm{sk}(B)\} \\ \textsc{In} & = & \{5\} \\ \textsc{Out} & = & \{8\} \\ \mathcal{S} & = & \{x_0 \stackrel{?}{=} A, x_1 \stackrel{?}{=} B, x_2 \stackrel{?}{=} \mathrm{pk}(A), x_3 \stackrel{?}{=} \mathrm{pk}(B), x_4 \stackrel{?}{=} \mathrm{sk}(B) \\ & & x_6 \stackrel{?}{=} \mathrm{ad}(x_5, x_4), x_7 \stackrel{?}{=} f(x_6), x_8 \stackrel{?}{=} \mathrm{ae}(x_7, x_2)\} \end{array}$$

The set of deduction states is $\{6, 7, 8\}$, there are no re-use states, the set of memory states is $\{0, \ldots, 4\}$ and the only reception state is 5. Assuming that the role B tests whether the received message is a cipher, one may add a ninth deduction state with $x_9 \stackrel{?}{=} \mathrm{ae}(x_6, x_3)$ and an equation $x_5 \stackrel{?}{=} x_9$.

In addition we assume that two symbolic derivations do not share any variable, and that equality between symbolic derivations is defined modulo a renaming of variables.

We represent graphically a symbolic derivation as follows:



- The sequence of variables $\mathcal{V}(1), \ldots, \mathcal{V}(n)$ represents the sequence $\mathcal{V}(\text{IND})$;
- an arrow pointing to $\mathcal{V}(i)$ means that $i \in \text{IN}$, as is the case for $\mathcal{V}(1)$ in the above figure;
- an arrow pointing away from $\mathcal{V}(i)$ means that $i \in \text{OUT}$, as is the case for $\mathcal{V}(n)$ in the above figure;
- $\mathcal{S}$ is the unification system.

For deduction systems of concern, closed derivations are such that each variable in $\mathcal{V}$ has a unique possible instance.

Lemma 6 *Let $\mathcal{I}$ be a subterm deduction system, and consider a closed and satisfiable $\mathcal{I}$-symbolic derivation $\mathcal{C} = (\mathcal{V}, \mathcal{S}, \mathcal{K}, \text{IN}, \text{OUT})$. Then there exists a unique ground substitution σ in normal form of support $\text{Im}(\mathcal{V})$ such that any unifier of $\mathcal{S}$ is an extension of σ.*

PROOF. Since the symbolic derivation $\mathcal{C} = (\mathcal{V}, \mathcal{S}, \mathcal{K}, \text{IN}, \text{OUT})$ is closed, it has by definition no input states, and thus all states are either knowledge, re-use or deduction states. By induction on the set of indices IND ordered by $<$:

Base case: Assume i is a minimal element in IND. By minimality i cannot be a re-use state. If it is a knowledge state then by definition there exists in $\mathcal{S}$ an equation $\mathcal{V}(i) \stackrel{?}{=} t$, with t a ground term in normal form, and thus for every unifier τ of $\mathcal{S}$ we must have $\mathcal{V}(i)\tau = t$. If i is a deduction state, and since it is minimal, the public symbol employed must be of arity 0 and hence is a constant, *i.e.*, again a ground term t. In both cases there exists a unique ground substitution σ in normal form defined on $\{\mathcal{V}(i)\}$ and such that any unifier of $\mathcal{S}$ is an extension of σ.

Induction case: Assume there exists a unique ground substitution σ in normal form with support: $\{\mathcal{V}(j) \mid j < i\}$ such that any unifier of $\mathcal{S}$ is an extension of σ. If i is a re-use state, we note that $\mathcal{V}(i)$ is already in the support of σ, and we are done. If it is a knowledge state, reasoning as in the basic case permits us to extend σ to $\mathcal{V}(i)$ if necessary. If it is a deduction state then there exists in $\mathcal{S}$ an equation $\mathcal{V}(i) \stackrel{?}{=} f(\mathcal{V}(j_1), \ldots, \mathcal{V}(j_n))$ with $j_1, \ldots, j_n < i$ that has to be satisfied by every unifier θ of $\mathcal{S}$. By induction every such unifier has to be equal to σ on $\{\mathcal{V}(j_1), \ldots, \mathcal{V}(j_n)\}$. Thus for every unifier θ of $\mathcal{S}$ we have $\mathcal{V}(i)\theta =_{\mathcal{E}} f(\mathcal{V}(j_1)\theta, \ldots, \mathcal{V}(j_n)\theta)$. By induction $f(\mathcal{V}(j_1)\theta, \ldots, \mathcal{V}(j_n)\theta) =_{\mathcal{E}} f(\mathcal{V}(j_1)\sigma, \ldots, \mathcal{V}(j_n)\sigma)$ and thus we must have $\mathcal{V}(i)\theta = (f(\mathcal{V}(j_1)\sigma, \ldots, \mathcal{V}(j_n)\sigma))\downarrow$. Therefore σ can be uniquely extended on $\mathcal{V}(i)$ by setting $\mathcal{V}(i)\sigma = (f(\mathcal{V}(j_1)\sigma, \ldots, \mathcal{V}(j_n)\sigma))\downarrow$ which is again a ground term.

□

By Lemma 6, if a derivation is closed, then for every $i \in \textsc{Ind}$ the variable $\mathcal{V}(i)$ is instantiated by a ground term. We say that a term t is *known at step* i in a closed symbolic derivation if there exists $j \leq i$ such that $\mathcal{V}(j)$ is instantiated by t.

Connection. We express the communication between two agents represented each by a symbolic derivation by *connecting* these symbolic derivations. This operation consists in identifying some input variables of one derivation with some output variables of the other and vice-versa. This connection should be compatible with the variable orderings inherited from each symbolic derivation, as detailed in the following definition:

Definition 3 *Let $\mathcal{C}_1$, $\mathcal{C}_2$ be two symbolic derivations with for $i \in \{1, 2\}$ $\mathcal{C}_i = (\mathcal{V}_i, \mathcal{S}_i, \mathcal{K}_i, \textsc{In}_i, \textsc{Out}_i)$, with disjoint sets of variables and index sets $(\textsc{Ind}_1, <_1)$ and $(\textsc{Ind}_2, <_2)$ respectively. Let I_1, I_2, be subsets of $\textsc{In}_1$, $\textsc{In}_2$, and O_1, O_2 be sub-multisets of $\textsc{Out}_1$, $\textsc{Out}_2$ respectively.*

Assume that there is a monotone bijection ϕ from $I_1 \cup I_2$ to $O_1 \cup O_2$ such that $\phi(I_1) = O_2$ and $\phi(I_2) = O_1$. A connection *of $\mathcal{C}_1$ and $\mathcal{C}_2$ over the* connection *function ϕ, denoted $\mathcal{C}_1 \circ_\phi \mathcal{C}_2$ is a symbolic derivation*

$$\mathcal{C} = (\mathcal{V}, \phi(\mathcal{S}_1 \cup \mathcal{S}_2), \mathcal{K}_1 \cup \mathcal{K}_2, (\textsc{In}_1 \cup \textsc{In}_2) \setminus (I_1 \cup I_2), (\textsc{Out}_1 \cup \textsc{Out}_2) \setminus (O_1 \cup O_2))$$

where:

- $(\textsc{Ind}, <)$ *is defined by:*
  - $\textsc{Ind} = (\textsc{Ind}_1 \setminus I_1) \cup (\textsc{Ind}_2 \setminus I_2)$;

– $<$ *is the transitive closure of the relation:* $<_1 \cup <_2$*;*

- ϕ *is extended to a renaming of variables in* $\mathrm{Var}(\mathcal{V}_1) \cup \mathrm{Var}(\mathcal{V}_2)$ *such that* $\phi(\mathcal{V}_1(i)) = \mathcal{V}_2(j)$ *(resp.* $\phi(\mathcal{V}_2(i)) = \mathcal{V}_1(j)$*) if* $i \in I_1$ *(resp.* I_2*) and* $\phi(i) = j$

When the exact connection function in a connection does not matter, is uniquely defined, or is described otherwise, we will omit the subscript and denote it $\mathcal{C}_1 \circ \mathcal{C}_2$*.*

A connection is *satisfiable* if the resulting symbolic derivation is satisfiable.

Example 5.3 *Let* $\mathcal{C}_h$ *be the symbolic derivation in Example 5.2:*

$$\begin{array}{rcl}
\textsc{Ind}_h & = & \{0, \ldots, 8\} \\
\mathcal{V}_h & = & i \in \textsc{Ind} \mapsto x_i \\
\mathcal{K}_h & = & \{A, B, \mathrm{pk}(A), \mathrm{pk}(B), \mathrm{sk}(B)\} \\
\textsc{In}_h & = & \{5\} \\
\textsc{Out}_h & = & \{0, 1, 2, 3, 4, 5, 6, 7, 8\} \\
\mathcal{S}_h & = & \{x_0 \stackrel{?}{=} A, x_1 \stackrel{?}{=} B, x_2 \stackrel{?}{=} \mathrm{pk}(A), x_3 \stackrel{?}{=} \mathrm{pk}(B), x_4 \stackrel{?}{=} \mathrm{sk}(B) \\
 & & x_6 \stackrel{?}{=} \mathrm{ad}(x_5, x_4), x_7 \stackrel{?}{=} f(x_6), x_8 \stackrel{?}{=} \mathrm{ae}(x_7, x_2)\}
\end{array}$$

We model the initial knowledge of the intruder with another symbolic derivation $\mathcal{C}_K$*:*

$$\begin{array}{rcl}
\textsc{Ind}_K & = & \left\{0^k, \ldots, 3^k\right\} \\
\mathcal{V}_K & = & i^k \in \textsc{Ind}_k \mapsto y_i \\
\mathcal{K}_K & = & \{A, B, \mathrm{pk}(A), \mathrm{pk}(B)\} \\
\textsc{In}_K & = & \emptyset \\
\textsc{Out}_K & = & \textsc{Ind}_K \\
\mathcal{S}_K & = & \left\{y_0 \stackrel{?}{=} A, y_1 \stackrel{?}{=} B, y_2 \stackrel{?}{=} \mathrm{pk}(A), y_3 \stackrel{?}{=} \mathrm{pk}(B)\right\}
\end{array}$$

and we let $\mathcal{C}'$ *be the following derivation:*

$$\begin{array}{rcl}
\textsc{Ind}' & = & \{0', \ldots, 8'\} \\
\mathcal{V}' & = & i' \in \textsc{Ind}' \mapsto z_i \\
\mathcal{K} & = & \{n\} \subset \mathrm{C}_{\mathrm{new}} \\
\textsc{In}' & = & \{0', 1', 2', 3', 8'\} \\
\textsc{Out}' & = & \{5'\} \cup \textsc{Ind}' \\
\mathcal{S}' & = & \{z_4 \stackrel{?}{=} n, z_5 \stackrel{?}{=} \mathrm{ae}(z_4, z_3), \\
 & & z_6 \stackrel{?}{=} f(z_4), z_7 \stackrel{?}{=} \mathrm{ae}(z_6, z_2), z_8 \stackrel{?}{=} z_7\}
\end{array}$$

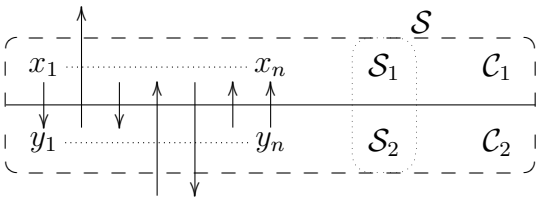
Let ϕ be the application from $0^k, 1^k, 2^k, 3^k, 5', 8$ to $0', 1', 2', 3', 5, 8'$ respectively and ψ be a function of empty domain. Then we have $(\mathcal{C}_h \circ_\psi \mathcal{C}_K) \circ_\phi \mathcal{C}'$:

$$
\begin{array}{rcl}
\textsc{Ind} & = & \left\{0, \ldots, 4, 0^k, \ldots, 3^k, 5', 6', 7', 6, 7, 8\right\} \\
\mathcal{V} & = & \mathcal{V}_h{}_{|\textsc{Ind}} \cup \mathcal{V}_K{}_{|\textsc{Ind}} \cup \mathcal{V}'{}_{|\textsc{Ind}} \\
\mathcal{K} & = & \{A, B, \mathrm{pk}(A), \mathrm{pk}(B), \mathrm{sk}(B), n\} \\
\textsc{In} & = & \emptyset \\
\textsc{Out} & = & \textsc{Ind} \cap \textsc{Ind}' \\
\mathcal{S} & = & \{x_0 \stackrel{?}{=} A, x_1 \stackrel{?}{=} B, x_2 \stackrel{?}{=} \mathrm{pk}(A), x_3 \stackrel{?}{=} \mathrm{pk}(B), x_4 \stackrel{?}{=} \mathrm{sk}(B) \\
 & & x_6 \stackrel{?}{=} \mathrm{ad}(x_5, x_4), x_7 \stackrel{?}{=} f(x_6), x_8 \stackrel{?}{=} \mathrm{ae}(x_7, x_2) \\
 & & y_0 \stackrel{?}{=} A, y_1 \stackrel{?}{=} B, y_2 \stackrel{?}{=} \mathrm{pk}(A), y_3 \stackrel{?}{=} \mathrm{pk}(B) \\
 & & z_5 \stackrel{?}{=} n, z_6 \stackrel{?}{=} \mathrm{ae}(z_5, z_3), \\
 & & z_7 \stackrel{?}{=} f(z_5), z_8 \stackrel{?}{=} \mathrm{ae}(z_7, z_2), z_9 \stackrel{?}{=} z_8\}
\end{array}
$$

with the ordering:

$$
\begin{array}{l}
0 < 1 < 2 < 3 < 4 < 5' < 6 < 7 < 8 \\
0^k < \ldots < 3^k < 4' < \ldots < 7' < 8
\end{array}
$$

The connection of two symbolic derivations $\mathcal{C}_1$ and $\mathcal{C}_2$ identifies variables in the input of one with variables in the output of the other. Variables that have been identified are removed from the input/output multiset of the resulting symbolic derivation $\mathcal{C}$. The set of equality constraints of $\mathcal{C}$ is the union of the equality constraints in $\mathcal{C}_1$ and $\mathcal{C}_2$, plus equalities stemming from the identification of input and output.



$\mathcal{C} = \mathcal{C}_1 \circ \mathcal{C}_2$

One easily checks that a connection of two symbolic derivations is also a symbolic derivation. Also, the associativity of function composition applied on the connections implies the associativity of the connection of symbolic derivations. Since connection functions are bijective, we will also identify $\mathcal{C} \circ \mathcal{C}'$ and $\mathcal{C}' \circ \mathcal{C}$. Thus when we compose several symbolic derivations, we will freely re-arrange or remove parentheses.

Traces. Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be two $\mathcal{I}$-symbolic derivations and φ be a connection such that $\mathcal{C} = \mathcal{C}_1 \circ_\varphi \mathcal{C}_2 = (\mathcal{V}, \mathcal{S}, \mathcal{K}, \textsc{In}, \textsc{Out})$ is closed. Assuming that $\mathcal{I}$ is

a subterm deduction system, Lemma 6 implies that there exists a unique ground substitution τ in normal form such that any unifier σ of $\mathcal{S}_1 \cup \mathcal{S}_2$ is equal to τ on the image of $\mathcal{V}$. We denote $\mathrm{Tr}_{\mathcal{C}_1 \circ_\varphi \mathcal{C}_2}(\mathcal{C}')$ the restriction of this substitution τ to the variables in the sequence of $\mathcal{C}'$, for $\mathcal{C}' \in \{\mathcal{C}_1, \mathcal{C}_2, \mathcal{C}_1 \circ_\varphi \mathcal{C}_2\}$, and call it the *trace* of the connection on $\mathcal{C}'$. In the remainder of this section we will always assume that trace substitutions are in normal form.

5.3.2 Solutions of symbolic derivations

Honest and attacker symbolic derivations We consider two types of symbolic derivations, one that is employed to model honest agents, and one to model an attacker.

Honest derivations. We do not impose constraints on the symbolic derivations representing honest principals, but for the avoidance of constants in $\mathrm{C}_{\mathrm{new}}$, since these constants are employed to model new values created by an attacker. We assume that nonces created by the honest agents are created at the beginning of their execution and are constants that are not in $\mathrm{C}_{\mathrm{new}}$.

Definition 4 *(Honest symbolic derivations) A symbolic derivation $\mathcal{C}$ is an* honest symbolic derivation *or HSD, if the constants appearing in $\mathcal{C}$ are not in* $\mathrm{C}_{\mathrm{new}}$.

Example 5.4 *The symbolic derivation for role B in Example 5.2 is honest.*

Attacker derivations. We consider an attacker modeled by a symbolic derivation in which only the following actions are possible:

- create a fresh, random value;
- receive from and send a message to one of the honest participant;
- deduce a new message from the set of already known messages;
- every state is in Out given that the intruder should be able to observe his own knowledge;
- given that we consider an actual execution, the set of states is totally ordered.

The definition of *attacker* symbolic derivations models these constraints:

Definition 5 *(Attacker symbolic derivations) A symbolic derivation $\mathcal{C} = (\mathcal{V}, \mathcal{S}, \mathcal{K}, \mathrm{In}, \mathrm{Out})$ is an* attacker symbolic derivation*, or ASD, if*

- IND *is a total order;*
- OUT *contains at least one occurrence of each index in* IND*;*
- $\mathcal{K}$ *is a subset of* $\mathrm{C}_{\mathrm{new}}$*, and*
- $\mathcal{S}$ *contains only equations of the form*

  Test equation: $\mathcal{V}(i) \stackrel{?}{=} \mathcal{V}(j)$ *for* $i, j \in$ IND*;*

  Deduction at state i**:** $\mathcal{V}(i) \stackrel{?}{=} f(\mathcal{V}(i_1), \ldots, \mathcal{V}(i_n))$*, with* $i_1, \ldots, i_n < i$*, and* f *a public symbol;*

  Nonce creation at state i**:** $\mathcal{V}(i) \stackrel{?}{=} c_i$ *with* $c_i \in \mathrm{C}_{\mathrm{new}}$.

The fact that the initial knowledge of the attacker is empty but for the nonces is not a restriction when analyzing protocols, as one can see from Example 5.3, and is justified in Section 5.3.3.

Example 5.5 *The following derivation* $\mathcal{C}'$ *is an ASD for the same deduction system as Example 5.2:*

$$\begin{array}{rcl}
\mathrm{IND}' & = & \{0', \ldots, 8'\} \\
\mathcal{V}' & = & i' \in \mathrm{IND}' \mapsto z_i \\
\mathcal{K} & = & \{n\} \subset \mathrm{C}_{\mathrm{new}} \\
\mathrm{IN}' & = & \{0', \ldots, 3', 8'\} \\
\mathrm{OUT}' & = & \{5'\} \cup \mathrm{IND}' \\
\mathcal{S}' & = & \{z_4 \stackrel{?}{=} n, z_5 \stackrel{?}{=} \mathrm{ae}(z_4, z_3), \\
 & & z_6 \stackrel{?}{=} f(z_4), z_7 \stackrel{?}{=} \mathrm{ae}(z_6, z_2), z_8 \stackrel{?}{=} z_7\}
\end{array}$$

Informally the ASD expresses that the attacker receives some key k*, creates a nonce* n*, sends the encrypted nonce to a role* B *as in Example 5.2. Then the attacker tries to check that applying* f *to* n *gives a term equal to the decryption of* B*'s response.*

Definition 6 *(Testing ASDs) An ASD is testing iff* $\mathcal{K}$ *is empty.*

Solutions of a symbolic derivation. Given a symbolic derivation $\mathcal{C}_h$ we denote $\mathcal{C}_h^\star$ the set of ASDs $\mathcal{C}$ such that $\mathcal{C}_h \circ \mathcal{C}$ is closed and satisfiable. In that case we say that $\mathcal{C}$ is a solution of $\mathcal{C}_h$.

Example 5.6 *In Example 5.3 the ASD* $\mathcal{C}'$ *is a solution of* $\mathcal{C}_h \circ \mathcal{C}_K$ *since* $(\mathcal{C}_h \circ_\psi \mathcal{C}_K) \circ_\phi \mathcal{C}'$ *has no input variables and* $\mathcal{S}$ *is satisfiable (by simply propagating the equalities* $x_0 = A, x_1 = B, \ldots$*).*

Decision problems

Satisfiability. Though it is expressed using different notations, the problem of the existence of a secrecy attack on a protocol execution with a finite number of messages is equivalent, in the setting of this section, to the satisfiability problem below. It has been shown to be NP-complete in [55] for the standard Dolev-Yao deduction system of Example 5.1.

$\mathcal{I}$**-Satisfiability**
Input: a HSD $\mathcal{C}$
Output: SAT iff $\mathcal{C}^\star \neq \emptyset$

Symbolic Equivalence. In this section, we are interested in the equivalence of HSDs *w.r.t.* an active intruder.

Definition 7 *Two HSDs* $\mathcal{C}_h$ *and* $\mathcal{C}'_h$ *are* symbolically equivalent *iff* $\mathcal{C}_h^\star = \mathcal{C}_h'^\star$.

Thanks to Lemma 7 one can easily see that when the states in the HSDs are totally ordered this notion is the same as the symbolic equivalence in [29].

$\mathcal{I}$**-Symbolic Equivalence**
Input: Two honest $\mathcal{I}$-symbolic derivations $\mathcal{C}_h$ and $\mathcal{C}'_h$
Output: SAT iff $\mathcal{C}_h{}^\star = \mathcal{C}_h'^\star$.

Remark. Let us remark that it makes sense to compare $\mathcal{C}_h^\star$ and $\mathcal{C}_h'^\star$ only if there exists a bijection between the in- and output states of these derivations such that every closed connection between an ASD and $\mathcal{C}_h$ can be mapped, using this bijection, to a closed connection between the same ASD and $\mathcal{C}'_h$. In order to simplify notations we implicitly quantify over all connection functions such that a composition is closed and satisfiable and consider the same connection (modulo the bijection) with the two HSDs $\mathcal{C}_h$ and $\mathcal{C}'_h$.

5.3.3 Relation with static equivalence

The problem we consider is whether two cryptographic processes, represented by HSDs in our setting, are observationally equivalent, in the sense that an attacker cannot build a sequence of interactions that would produce different results when applied to the two processes. Solving this problem has many applications. For instance if the two processes only differ by a data value this

shows that this data is confidential. In [2] the observational equivalence problem for an attacker who does not interact with the honest agents is reduced to the one of the *static equivalence* between two sequences of messages.

In the broader setting in which an attacker interacts online with the honest participants, [35] reduces the observational equivalence to trace equivalence for a class of processes corresponding to honest symbolic derivations. Their trace equivalence corresponds to symbolic equivalence in our setting.

Static equivalence.

Contexts. Let us first recall the notion of static equivalence between frames as introduced in [2]. A *frame* is a substitution σ of finite support $\{x_1, \ldots, x_n\}$ hiding a finite sequence $\vec{c}$ of constants, which is denoted $\nu\vec{c}\cdot\sigma$. A *public constructor* is a function symbol f of arity k such that, if the intruder knows $t_1, \ldots, t_k$ he also knows $f(t_1, \ldots, t_k)$. A public context M over the frame $\nu\vec{c}\cdot\sigma$ is a term whose variables are in the support of σ, whose constants are not in $\vec{c}$ and whose other symbols are public constructors. Finally, equality is defined modulo an equational theory $\mathcal{E}$.

Constants. Without loss of generality, we can assume that all free constants in a context M are distinct from those appearing in σ: the rationale for this is that if a free constant c_0 is in σ but not in $\vec{c}$ we can always consider the public contexts on the frame $\nu c, \vec{c}_0 \cdot \{x_0 \mapsto c\} \cup \sigma$ which are the same—but for the replacement of c by x_0—as those on the frame $\nu\vec{c}\cdot\sigma$. This motivates the splitting of the set of free constants into two sets, C and $\mathrm{C_{new}}$, where C designates those free constants that can be used by honest users, and $\mathrm{C_{new}}$ those that can be used by an attacker. We emphasize here that, as in [2], the attacker can manipulate terms containing constants in C. We have just ensured that these constants have to be passed explicitely to the attacker through the substitution σ. When considering symbolic derivations, this translates into imposing that the knowledge of an ASD must contain only constants in $\mathrm{C_{new}}$.

Let us now recast the definition of static equivalence, as stated in [2], according to these assumptions.

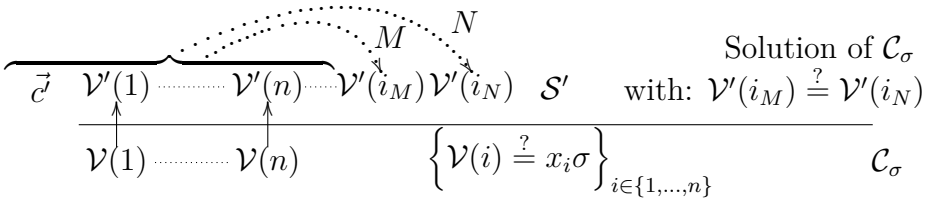
Definition 8 *(Static equivalence) Two frames $\varphi = \nu\vec{c}\cdot\sigma$ and $\psi = \nu\vec{c'}\cdot\tau$ that have the same domain are statically equivalent if for any public contexts M and N whose constants are not in $\vec{c}\cup\vec{c'}$ one has $M\sigma =_{\mathcal{E}} N\sigma$ iff one has $M\tau =_{\mathcal{E}} N\tau$.*

The definition of *contexts* corresponds to the notion of derivation in the following sense: let $\mathcal{I}$ to be the deduction system defined over a signature $\mathcal{F}$, modulo an equational theory $\mathcal{E}$, with $\mathcal{P}$ equal to the set of public symbols. We note that, given the possible deductions, the quantification is over all symbolic derivations that takes in input terms in the frame and constants not belonging to these frames, and thus in $\mathrm{C}_{\mathrm{new}}$. Static equivalence states that any couple (M, N) of contexts yields the same result in one frame iff it yields the same result in the other frame. This suggests us to express static equivalence of frames in terms of sets of solutions of symbolic derivations as follows.

First, to a substitution σ of finite support $x_1, \ldots, x_n$ we associate the closed symbolic derivation:

$$\mathcal{C}_\sigma = (\mathcal{V}, \left\{\mathcal{V}(i) \stackrel{?}{=} x_i\sigma\right\}_{i=1,\ldots,n}, \mathrm{Im}(\sigma), \emptyset, \{1, \ldots, n\})$$

with $\mathcal{V}$ of support $\{1, \ldots, n\}$. To represent the construction of contexts by the attacker, we consider symbolic derivations $\mathcal{C}_\mathcal{I} = (\mathcal{V}_\mathcal{I}, \mathcal{S}_\mathcal{I}, \vec{c_\mathcal{I}}, \mathrm{In}_\mathcal{I}, \emptyset)$, with $|\mathrm{In}_\mathcal{I}| = n$, and $\vec{c_\mathcal{I}}$ a finite subset of $\mathrm{C}_{\mathrm{new}}$. The equality of two contexts M and N over σ can then be translated as the satisfiability of the following composition of symbolic derivations:



Clearly, two frames $\nu\vec{c} \cdot \sigma$ and $\nu\vec{c} \cdot \tau$ are statically equivalent, with the standard definition, iff for any ASD $\mathcal{C}'$, $\mathcal{C}' \circ \mathcal{C}_\sigma$ is closed and satisfiable iff $\mathcal{C}' \circ \mathcal{C}_\tau$ is closed and satisfiable. In our notation this is translated into the equality $\mathcal{C}_\sigma^\star = \mathcal{C}_\tau^\star$, and the problem of deciding whether two closed frames are in static equivalence is the same problem as deciding whether two closed symbolic derivations are symbolically equivalent.

Equational theories and equivalence The original problem of interest considered is whether two cryptographic processes are bisimilar for an external observer. In [2] this problem is reduced to the one of the static equivalence between two sequences of ground messages. However, the cryptographic operations considered were total, which means *e.g.*, that a decryption applied on a message with a key always returns a message even when the decryption key does not match the encryption key. As a result, the observer is not aware

of whether a cryptographic operation is successful. We note that under these assumptions the frames:

$$\begin{cases} \varphi &= \nu a, k \cdot \{x_1 \mapsto enc(a,k), x_2 \mapsto k^{-1}\} \\ \psi &= \nu a, k', k \cdot \{x_1 \mapsto enc(a,k'), x_2 \mapsto k^{-1}\} \end{cases}$$

are equivalent when assuming that an observer has no way to differentiate $a =_{\mathcal{E}} dec(x_1, x_2) \cdot \varphi$ and $dec(enc(a, k'), k^{-1}) = dec(x_1, x_2) \cdot \psi$. This is the case *e.g.*, when no padding nor other security measure permits one to check that the decryption has succeeded. But when one assumes that the cryptographic primitives abstracted by the *enc* and *dec* symbols are such that $dec(enc(a, k'), k^{-1})$ can be detected to be an incorrect decryption result (for example because it does not have a correct padding), the two frames φ and ψ shall be distinguishable. The choice between the two models shall be made on a *per operation* basis and affects both the HSDs and the ASDs:

HSDs: In the second case, it makes sense to assume that there is no "decomposition" symbol in the honest symbolic derivations considered (assuming thereby that in a prudent implementation a raised exception would have stopped the execution), while in the first case this distinction is irrelevant.

ASDs: In the second case, we have to ensure that the traces seen by the intruder are equivalent *w.r.t.* to equational rules applied on the contexts constructed by the intruder, *i.e.* we have to ensure that the unification system is normalized in the same way when composing an ASD with two HSDs. Technically, this amounts to a guessing of a set of narrowing steps (see below) on the unification system of an ASD *before* composing it with the HSDs. In the first case, one does not guess the normalization steps before composing, and just relies on the satisfiability of the unification system.

In the rest of this section we consider only the case of silently failing operations, given that there is no conceptual difficulty to change the proofs and algorithms so that they are in accordance with the first case. We outline in Section 5.4.6 how to adapt the decision problem and the resolution algorithm proposed for the silently failing case.

5.4 The case of a subterm deduction system

This subsection is devoted to the proof of the main theorem of this section.

Theorem 1 *Symbolic equivalence is decidable for subterm deduction systems.*

In order to prove this theorem it suffices to provide a decision procedure for the inclusion $\mathcal{C}_h^\star \subseteq \mathcal{C}_h'^\star$. We first prove in Section 5.4.2 that this inclusion is equivalent to the inclusion $\mathcal{C}_h^{\text{sf}+} \subseteq \mathcal{C}_h'^\star$ where ASDs in $\mathcal{C}_h^{\text{sf}+}$ are the connections between the stutter free ASDs (formally defined in Section 5.4.1) in $\mathcal{C}_h^\star$ and an ASD of bounded size. Since this inclusion is hard to test directly we introduce in Section 5.4.3 another set $\text{Sol}(\mathcal{C}_h)$ such that $\mathcal{C}_h^{\text{sf}+} \subseteq \text{Sol}(\mathcal{C}_h) \subseteq \mathcal{C}_h^\star$ and thus reduce the original inclusion problem to the problem of deciding whether $\text{Sol}(\mathcal{C}_h) \subseteq \mathcal{C}_h'^\star$. In Section 5.4.4 we introduce a well-founded ordering on ASDs and prove that it suffices to decide this inclusion for the minimal ASDs in $\text{Sol}(\mathcal{C}_h)$. We finally prove in Section 5.4.5 that these minimal ASDs have a polynomial size *w.r.t.* the size of $\mathcal{C}_h$. This leads to our non-deterministic algorithm that guesses an ASD of a size within the bound, tests whether this ASD is in $\text{Sol}(\mathcal{C}_h)$ and, if it is the case, tests whether it is in $\mathcal{C}_h'^\star$. We discuss the complexity of this algorithm in Section 5.4.6.

5.4.1 (De)composition rules and stutter free derivations

Decomposition rules. Let $\mathcal{C}_h, \mathcal{C}_h'$ be two HSDs and consider an attacker derivation $\mathcal{C}_\mathcal{I} = (\mathcal{V}_\mathcal{I}, \mathcal{S}_\mathcal{I}, \mathcal{K}_\mathcal{I}, \text{In}_\mathcal{I}, \text{Out}_\mathcal{I})$ in $\mathcal{C}_h^\star$, and let $\sigma = \text{Tr}_{\mathcal{C}_h \circ \mathcal{C}_\mathcal{I}}(\mathcal{C}_\mathcal{I})$. Let $i_0 \in \text{Ind}_\mathcal{I}$ be a deduction state, and assume that the deduction rule is applied with the equation $\mathcal{V}_\mathcal{I}(i_0) \stackrel{?}{=} f(\mathcal{V}_\mathcal{I}(i_1), \ldots, \mathcal{V}_\mathcal{I}(i_n))$. If $f(\mathcal{V}_\mathcal{I}(i_1)\sigma, \ldots, \mathcal{V}_\mathcal{I}(i_n)\sigma)$ is in normal form we say that the deduction state i_0 is a *composition state*. Otherwise, we say that the deduction state i_0 is a *decomposition state*. Note that in that case $\mathcal{V}_\mathcal{I}(i_0)\sigma$ is a subterm of a previously known term at some state $i < i_0$. We remark that if $\mathcal{C}_\mathcal{I} \in \mathcal{C}_h^\star \cap \mathcal{C}_h'^\star$:

- the deduction states of $\mathcal{C}_\mathcal{I}$ do not depend on whether we are considering $\mathcal{C}_\mathcal{I} \circ \mathcal{C}_h$ or $\mathcal{C}_\mathcal{I} \circ \mathcal{C}_h'$;
- but the composition and decomposition states may be different.

Deductions and symbolic derivations. Given a public symbol f we call *symbolic deduction* and denote $\mathcal{D}_f$ the symbolic derivation:

$$(\mathcal{V}_f, \left\{\mathcal{V}(n+1) \stackrel{?}{=} f(\mathcal{V}(1), \ldots, \mathcal{V}(n))\right\}, \emptyset, \{1, \ldots, n\}, \{1, \ldots, n, n+1, n+1\})$$

where $\mathcal{V}_f : \{1, \ldots, n+1\} \to \mathcal{X}$ is an injective function. The order $<$ on $\{1, \ldots, n+1\}$ is such that for all $i \in \{1, \ldots, n\}$ we have $i < n+1$.

Stutter free symbolic derivations. Let us now define *stutter free* symbolic derivations, which correspond intuitively to symbolic derivations which only solve reachability problems.

Let $\mathcal{C}_\mathcal{I} = (\mathcal{V}_\mathcal{I}, \mathcal{S}_\mathcal{I}, \mathcal{K}_\mathcal{I}, \text{IN}_\mathcal{I}, \text{OUT}_\mathcal{I})$ be an ASD in $\mathcal{C}_h^\star$. We say that $\mathcal{C}_\mathcal{I}$ is *aware* with respect to $\mathcal{C}_h$ if whenever i, j are two deduction or memory states of $\mathcal{C}_\mathcal{I}$ such that $\mathcal{V}_\mathcal{I}(i)\theta =_\mathcal{E} \mathcal{V}_\mathcal{I}(j)\theta$, with $\theta = \text{Tr}_{\mathcal{C}_\mathcal{I} \circ \mathcal{C}_h}(\mathcal{C}_\mathcal{I})$ then $\mathcal{S}_\mathcal{I}$ contains the equation $\mathcal{V}_\mathcal{I}(i) \stackrel{?}{=} \mathcal{V}_\mathcal{I}(j)$.

Definition 9 *(stutter free derivation) Let $\mathcal{C}_\mathcal{I} = (\mathcal{V}_\mathcal{I}, \mathcal{S}_\mathcal{I}, \mathcal{K}_\mathcal{I}, \text{IN}_\mathcal{I}, \text{OUT}_\mathcal{I}) \in \mathcal{C}_h^\star$ be an ASD. We say that $\mathcal{C}_\mathcal{I}$ is* stutter free *if:*

- *There exists a most general unifier θ of $\mathcal{S}_\mathcal{I}$* in the empty theory*;*
- *Whenever $i \neq j$ for non-reuse states i, j we have $\mathcal{V}_\mathcal{I}(i)\theta \neq_\mathcal{E} \mathcal{V}_\mathcal{I}(j)\theta$;*
- *For every couple of memory or deduction states i, j with $j < i$ we have $\mathcal{V}(j)\sigma \neq \mathcal{V}(i)\sigma$, where $\sigma = \text{Tr}_{\mathcal{C}_\mathcal{I} \circ \mathcal{C}_h}(\mathcal{C}_\mathcal{I})$.*

The third point of the definition implies that no action of the attacker will produce a value that was hitherto known, hence the name of stutter free derivations. The first two points enforce that in a stutter free derivation the attacker does not test the messages he receives or produces. The first point forbids *implicit* tests in which some equality from the equational theory has to be satisfied whereas the second point forbids an explicit testing between the values of two states. Finally let us note that when considering the connection with a given HSD, every stutter free ASD is aware.

Proposition 2 *Let $\mathcal{C}_\mathcal{I} = (\mathcal{V}_\mathcal{I}, \mathcal{S}_\mathcal{I}, \mathcal{K}_\mathcal{I}, \text{IN}_\mathcal{I}, \text{OUT}_\mathcal{I}) \in \mathcal{C}_h^\star$ be a stutter free ASD. We have:*

(i) $\mathcal{V}_\mathcal{I}$ is injective on non-reuse states;

(ii) For any ground substitution σ of domain $\text{IN}_\mathcal{I}$ the unification system $\mathcal{S}_\mathcal{I}\sigma$ is satisfiable in the empty theory.

PROOF. (i) Using the notations of Definition 9 if there exists $i \neq j$ where neither i nor j are reuse states such that $\mathcal{V}_\mathcal{I}(i) = \mathcal{V}_\mathcal{I}(j)$ then we have $\mathcal{V}_\mathcal{I}(i)\theta = \mathcal{V}_\mathcal{I}(j)\theta$, and thus $\mathcal{C}_\mathcal{I}$ is not stutter free.

(ii) We recall that a unification system $\mathcal{S}_\mathcal{I}$ is in solved form in the empty theory if and only if there exists an ordering $<_u$ on variables such that $\mathcal{S}_\mathcal{I}$ contains, for each variable x, at most one equation $x \stackrel{?}{=} t$ and if for every $y \in \text{Var}(t)$ we have $y <_u x$. First let us notice that since $\mathcal{C}_\mathcal{I}$ is stutter free, $\mathcal{S}_\mathcal{I}$ does not contain any test equation (for the second condition would otherwise be impossible to satisfy for any unifier of $\mathcal{S}_\mathcal{I}$.) Thus $\mathcal{S}_\mathcal{I}$ contains exactly one equation $\mathcal{V}_\mathcal{I}(i) \stackrel{?}{=} t$ if i is not an

input or the re-use of an input state, and none otherwise. In the former case we can assume that for a mgu θ of $\mathcal{S}$ we have $\mathcal{V}(i)\theta = \mathcal{V}(i)$. Given the condition on the deduction equations, $\mathcal{S}_\mathcal{I}$ is in solved form, adding to $\mathcal{S}_\mathcal{I}$ equations $\mathcal{V}_\mathcal{I}(i) \stackrel{?}{=} t_i$, for $i \in \text{In}_\mathcal{I}$ and t_i a ground term thus leads to a unification system also in solved form.

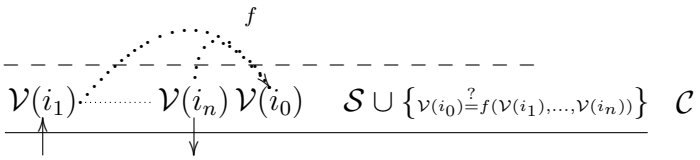
□

Notations. In the rest of this section, when one considers an ASD in $\mathcal{C}_h^\star$, and unless otherwise specified, decomposition and composition states and stutter freeness are considered *w.r.t.* the composition with $\mathcal{C}_h$. Given an HSD $\mathcal{C}_h$, we denote $\mathcal{C}_h^{\text{sf}}$ the subset of stutter free symbolic derivations in $\mathcal{C}_h^\star$.

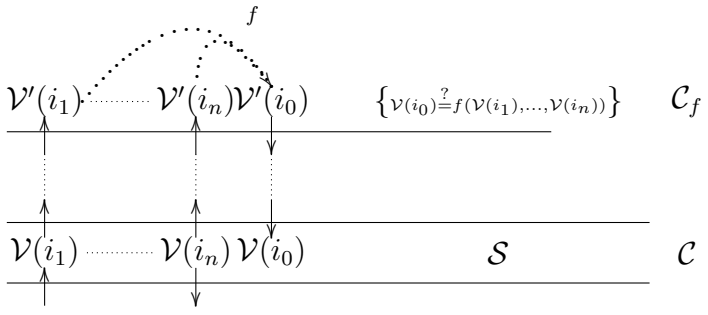
Proposition 3 *Let $\mathcal{C}_h$ be a HSD. Then for any $\mathcal{C}_\mathcal{I} \in \mathcal{C}_h^\star$ there exist two ASDs $\mathcal{C}_d$ and $\mathcal{C}_c$ such that*

- *There exists ψ such that $\mathcal{C}_d \circ_\psi \mathcal{C}_c = \mathcal{C}_\mathcal{I}$;*
- *For any satisfiable connection φ between $\mathcal{C}_\mathcal{I}$ and $\mathcal{C}_h$ we have in $(\mathcal{C}_c \circ_\psi \mathcal{C}_d) \circ_\varphi \mathcal{C}_h$:*
  - *all deduction states of $\mathcal{C}_d$ are decompositions and $\mathcal{C}_d$ does not contain any memory state,*
  - *all deduction states of $\mathcal{C}_c$ are composition states.*
- *Furthermore, if $\mathcal{C}_\mathcal{I}$ is in $\mathcal{C}_h^{\text{sf}}$ then $\mathcal{C}_c \in (\mathcal{C}_d \circ \mathcal{C}_h)^{\text{sf}}$.*

Proof. Consider a satisfiable connection $\mathcal{C} = \mathcal{C}_h \circ \mathcal{C}_\mathcal{I}$ between $\mathcal{C}_h$ and $\mathcal{C}_\mathcal{I}$. We construct $\mathcal{C}_d$ by removing deductions from $\mathcal{C}$ in the following way. Assume i_0 is a decomposition state when connecting $\mathcal{C}_\mathcal{I}$ with $\mathcal{C}_h$:



We replace the deduction with a connection with a symbolic deduction as follows:



We let $\mathcal{C}_d$ be the connection between all the extracted symbolic deductions. It is clear that the derivation $\mathcal{C}_\mathcal{I}$ in which all the decomposition states have been removed is now a symbolic derivation in which all deduction states are composition states.

To finish the proof of the proposition, let us now assume that $\mathcal{C}_\mathcal{I}$ is stutter free. Since we have not introduced any equality between variables in the process of removing decompositions, nor introduced any new deduction or memory state. The resulting symbolic derivation is thus also stutter free when composed with $\mathcal{C}_d \circ \mathcal{C}_h$. □

5.4.2 Reduction of $\mathcal{C}_h^\star \subseteq \mathcal{C}_h'^\star$ to $\mathcal{C}_h^{\text{sf}+} \subseteq \mathcal{C}_h'^\star$

We first prove that every ASD can be written as the connection between a stutter free ASD and a testing ASD in which no new term is deduced (Lemma 7). This implies the reduction of the inclusion problem to the one of checking whether, for any stutter free ASD in $\mathcal{C}_h^\star$, the connections of this ASD with $\mathcal{C}_h$ and $\mathcal{C}_h'$ result in *closed* symbolic derivations $\mathcal{C}_1$ and $\mathcal{C}_2$ such that $\mathcal{C}_1^\star \subseteq \mathcal{C}_2^\star$ (Lemma 8). Given a stutter free ASD in $\mathcal{C}_h^\star$ this latter test is simple since it suffices to consider the connection with ASD that have at most one deduction (Propositions 4, 5).

Lemma 7 *Let $\mathcal{C}_h$ be a HSD. Then for every $\mathcal{C}_\mathcal{I}$ in $\mathcal{C}_h^\star$ there exist two ASDs $\mathcal{C}' = (\mathcal{V}', \mathcal{S}', \mathcal{K}', \text{In}', \text{Out}')$ and $\mathcal{C}_t = (\mathcal{V}_t, \mathcal{S}_t, \mathcal{K}_t, \text{In}_t, \text{Out}_t)$ such that:*

- *$\mathcal{C}'$ is in $\mathcal{C}_h^{\text{sf}}$ and $\mathcal{C}_t$ is testing;*
- *$\{\mathcal{V}_t(i)\text{Tr}_{\mathcal{C}_t \circ \mathcal{C}' \circ \mathcal{C}_h}(\mathcal{C}_t)\}_{i \in \text{Ind}_t} \subseteq \{\mathcal{V}'(i)\text{Tr}_{\mathcal{C}' \circ \mathcal{C}_h}(\mathcal{C}')\}_{i \in \text{Ind}'}$;*
- *For every HSD $\mathcal{C}_h'$, if $\mathcal{C}_\mathcal{I}$ is aware in its composition with $\mathcal{C}_h$, then $\mathcal{C}' \circ \mathcal{C}_t \in \mathcal{C}_h'^\star$ iff $\mathcal{C}_\mathcal{I} \in \mathcal{C}_h'^\star$*

Proof. Let $\sigma = \text{Tr}_{\mathcal{C}_h \circ \mathcal{C}_\mathcal{I}}(\mathcal{C}_\mathcal{I})$ and D be the set of deduction or memory states in $\text{Ind}_\mathcal{I}$. We define $\psi : \text{Ind}_I \to \text{Ind}_I$ an application such that for any state $i \in D$ we have $\psi(i) = \min\{j \leq i \mid j \in D \text{ and } \mathcal{V}(j)\sigma = \mathcal{V}(i)\sigma\}$, and ψ is the identity on the states in $\text{Ind}_\mathcal{I} \setminus D$. Let $\theta : \mathcal{V}_I(i) \mapsto \mathcal{V}_I(\psi(i))$. Let us construct $\mathcal{C}'$ and $\mathcal{C}_t$:

Internal states: $\text{Ind}' = \psi(\text{Ind}_\mathcal{I})$, $\text{Ind}_t = \text{Ind}_I$;

Variables: $\mathcal{V}_t = \mathcal{V}_\mathcal{I}$ and $\mathcal{V}' = \mathcal{V}_{\mathcal{I}|\text{Ind}'}$;

Unification systems: Let $\mathcal{S}_0$ be the set of equations that are deductions in $\mathcal{C}_\mathcal{I}$ for some state $i \in \text{Ind}'$. Then we define $\mathcal{S}' = \mathcal{S}_0\theta$ and $\mathcal{S}_t = \mathcal{S}_I \setminus \mathcal{S}_0$;

Knowledge: $\mathcal{K}' = \mathcal{K}_{\mathcal{I}}$ and $\mathcal{K}_t = \emptyset$;

Input states: Any state in $\textsc{Ind}' \subseteq \textsc{Ind}_{\mathcal{I}}$ which is not a deduction state in $\mathcal{C}_t$ is an input state of $\mathcal{C}_t$. Input states of $\mathcal{C}'$ are the same as the ones in $\mathcal{C}_{\mathcal{I}}$;

Output states: $\textsc{Out}_t = \textsc{Ind}_t$ and $\textsc{Out}' = \textsc{Out}_{\mathcal{I}} \cup \textsc{Ind}'$.

We define the connection ϕ to be the identity mapping from $\textsc{In}_t$ to $\textsc{Out}'$.

This construction keeps the first deductions of a term in $\mathcal{C}'$ and records the redundant deductions by adding the deduction equations in $\mathcal{C}_t$. The application of θ on $\mathcal{S}'$ ensures that $\mathcal{C}'$ still is a symbolic derivation even though some deduction states were removed. However we have changed in this transformation the deductions by considering equalities induced by the connection with $\mathcal{C}_h$. The awareness of $\mathcal{C}_{\mathcal{I}}$ in its connection with $\mathcal{C}_h$ implies that these equalities hold for every satisfiable connection of $\mathcal{C}_{\mathcal{I}}$ with another $\mathcal{C}'_h$.

The first point is true by construction. The proof of the second point also follows directly from this construction since, by induction on $\textsc{Ind}'$ (resp. on $\textsc{Ind}_t$) we can prove that for all $i \in \textsc{Ind}'$ (resp. for all $i \in \textsc{Ind}_t$) $\mathcal{V}'(i)\mathrm{Tr}_{\mathcal{C}'\circ\mathcal{C}_h}(\mathcal{C}') = \mathcal{V}_{\mathcal{I}}(i)\sigma$ (resp. $\mathcal{V}_t(i)\mathrm{Tr}_{\mathcal{C}_t\circ_\phi\mathcal{C}_t(\mathcal{C}'\circ\mathcal{C}_h)}(\mathcal{C}_t) = \mathcal{V}_{\mathcal{I}}(i)\sigma$). For the third point we employ awareness to prove that the set of substitutions satisfying $\mathcal{S}_{\mathcal{I}}$ is equal (modulo a renaming of variables) to the set of substitutions satisfying the unification system of $\mathcal{C}' \circ \mathcal{C}_t$. □

We note that $\mathcal{C}'$ has the same input states as $\mathcal{C}_{\mathcal{I}}$. Thus, if there exists a closed connection between $\mathcal{C}_{\mathcal{I}}$ and a HSD $\mathcal{C}'_h$ then the same connection function can be employed to obtain a closed connection between $\mathcal{C}'$ and $\mathcal{C}_h$.

Lemma 8 *Let $\mathcal{C}_h$ and $\mathcal{C}'_h$ be two HSDs. We have $\mathcal{C}_h^\star \subseteq \mathcal{C}_h'^\star$ if, and only if:*

- $\mathcal{C}_h^{\mathrm{sf}} \subseteq \mathcal{C}_h'^\star$;
- *and for each aware ASD $\mathcal{C}_{\mathcal{I}} \in \mathcal{C}_h^{\mathrm{sf}}$ and for all testing ASD $\mathcal{C}_t \in (\mathcal{C}_{\mathcal{I}} \circ \mathcal{C}_h)^\star$ we have $\mathcal{C}_t \in (\mathcal{C}_{\mathcal{I}} \circ \mathcal{C}'_h)^\star$.*

Proof. Let us first prove the direct implication. Let us assume that $\mathcal{C}_h^\star \subseteq \mathcal{C}_h'^\star$. By definition we then have $\mathcal{C}_h^{\mathrm{sf}} \subseteq \mathcal{C}_h'^\star$. By contradiction let us assume that there exists $\mathcal{C} \in \mathcal{C}_h^{\mathrm{sf}}$ such that $\mathcal{C}_1 = \mathcal{C} \circ \mathcal{C}_h$ and $\mathcal{C}_2 = \mathcal{C} \circ \mathcal{C}'_h$ are such that there exists a testing ASD $\mathcal{C}_t$ in $\mathcal{C}_1^\star \not\subseteq \mathcal{C}_2^\star$. By construction $\mathcal{C} \circ \mathcal{C}_t$ is an ASD in $\mathcal{C}_h^\star \setminus \mathcal{C}_h'^\star$.

Let us prove the converse direction by contra-positive reasoning. Assume that $\mathcal{C}_h^\star \setminus \mathcal{C}_h'^\star \neq \emptyset$ and thus contains an ASD $\mathcal{C}_{\mathcal{I}}$. Adding test equalities to

$\mathcal{C}_\mathcal{I}$ satisfied by its connection with $\mathcal{C}_h$, we assume *wlog* that $\mathcal{C}_\mathcal{I}$ is aware. Let $\mathcal{C}', \mathcal{C}_t$ be the ASDs obtained by applying Lemma 7 on $\mathcal{C}_\mathcal{I}$ *w.r.t.* $\mathcal{C}_h$. Since $\mathcal{C}_\mathcal{I} \circ \mathcal{C}'_h$ is not satisfiable, then neither is $(\mathcal{C}'_h \circ \mathcal{C}') \circ \mathcal{C}_t$. Thus either $\mathcal{C}'_h \circ \mathcal{C}'$ is not satisfiable, or it is satisfiable, but $(\mathcal{C}'_h \circ \mathcal{C}') \circ \mathcal{C}_t$ is not. In the first case we have by definition of $\mathcal{C}'$ that $\mathcal{C}_h^{\text{sf}} \not\subseteq \mathcal{C}_h'^{\star}$. In the second case we have found an ASD $\mathcal{C}'$ in $\mathcal{C}_h^{\text{sf}}$ such that $\mathcal{C}' \circ \mathcal{C}_h$ and $\mathcal{C}' \circ \mathcal{C}'_h$ are satisfiable closed derivations and $(\mathcal{C}' \circ \mathcal{C}_h)^\star \not\subseteq (\mathcal{C}' \circ \mathcal{C}'_h)^\star$. □

Let us assume that we are given two HSDs $\mathcal{C}_h$ and $\mathcal{C}'_h$ such that $\mathcal{C}_h^{\text{sf}} \subseteq \mathcal{C}_h'^{\star}$. Our goal is to show that $\mathcal{C}_h^\star \subseteq \mathcal{C}_h'^{\star}$. Given an ASD $\mathcal{C}_\mathcal{I} \in \mathcal{C}_h^{\text{sf}}$ we define

$$\chi(\mathcal{C}_\mathcal{I}) = \left\{ \mathcal{C}_t \text{ testing ASD} \,|\, \mathcal{C}_t \circ \mathcal{C}_\mathcal{I} \in \mathcal{C}_h^\star \setminus \mathcal{C}_h'^{\star} \right\}$$

Intuitively this is the set of testing ASDs that permit one to distinguish $\mathcal{C}_h$ from $\mathcal{C}'_h$. By Lemma 8, $\mathcal{C}_h^\star \not\subseteq \mathcal{C}_h'^{\star}$ if, and only if, there exists an ASD $\mathcal{C}_\mathcal{I}$ such that $\chi(\mathcal{C}_\mathcal{I}) \neq \emptyset$.

Proposition 4 *$\mathcal{C}_h^\star \not\subseteq \mathcal{C}_h'^{\star}$ if, and only if, there exists $\mathcal{C}_\mathcal{I} \in \mathcal{C}_h^{\text{sf}}$ such that $\chi(\mathcal{C}_\mathcal{I})$ contains an ASD $\mathcal{C}_t$ with at most one deduction and one equality test.*

Proof. The converse direction is trivial.

First let us note that if $\mathcal{C}' \in \mathcal{C}_h^\star \setminus \mathcal{C}_h'^{\star}$ then adding test equations to $\mathcal{C}'$ which are satisfied by $\text{Tr}_{\mathcal{C}' \circ \mathcal{C}_h}(\mathcal{C}')$ yields another symbolic derivation in $\mathcal{C}' \in \mathcal{C}_h^\star \setminus \mathcal{C}_h'^{\star}$. Thus and *wlog* we let $\mathcal{C}' \in \mathcal{C}_h^\star \setminus \mathcal{C}_h'^{\star}$ be an aware ASD. According to Lemma 7 $\mathcal{C}'$ can be split into one stutter free derivation $\mathcal{C}_\mathcal{I} = (\mathcal{V}_\mathcal{I}, \mathcal{S}_\mathcal{I}, \mathcal{K}_\mathcal{I}, \text{In}_\mathcal{I}, \text{Out}_\mathcal{I})$ and one test derivation $\mathcal{C}_t = (\mathcal{V}_t, \mathcal{S}_t, \mathcal{K}_t, \text{In}_t, \text{Out}_t)$. We also define a partition $\mathcal{S}_t^d \cup \mathcal{S}_t^t$ of $\mathcal{S}_t$ such that $\mathcal{S}_t^d$ contains only deduction equations and $\mathcal{S}_t^t$ contains only test equations. Let $\mathcal{C}_t^d = (\mathcal{V}_t, \mathcal{S}_t^d, \mathcal{K}_t, \text{In}_t, \text{Out}_t)$. Let us define the following substitutions:

$$\begin{cases} \sigma_\mathcal{I} = \text{Tr}_{\mathcal{C}_\mathcal{I} \circ \mathcal{C}_h}(\mathcal{C}_\mathcal{I}) & \sigma'_\mathcal{I} = \text{Tr}_{\mathcal{C}_\mathcal{I} \circ \mathcal{C}'_h}(\mathcal{C}_\mathcal{I}) \\ \sigma_t = \text{Tr}_{\mathcal{C}_t \circ \mathcal{C}_\mathcal{I} \circ \mathcal{C}_h}(\mathcal{C}_t) \end{cases}$$

We also define a substitution σ'_t on $\mathcal{V}(\text{Ind}_t)$ in the following way. First we note that, if $\mathcal{V}_t(i) = \mathcal{V}_t(j)$ for two distinct states i, j which are not reuse states, we can introduce a new variable x, change $\mathcal{V}_t(j)$ to x, and introduce in $\mathcal{S}_t$ a new test equation $\mathcal{V}_t(i) \stackrel{?}{=} x$. In other words we can assume *wlog* that $\mathcal{V}_t$ is injective on states which are not reuse states. This permits one to ensure that the subset $\mathcal{S}_t^d$ of equations which are not test equations is satisfiable in any closed connection with another symbolic derivation. We define $\sigma_t^d = \text{Tr}_{\mathcal{C}_t^d \circ \mathcal{C}_\mathcal{I} \circ \mathcal{C}'_h}(\mathcal{C}_t^d)$.

By the second point of Lemma 7 there exists a mapping $\psi : \text{Ind}_t \to \text{Ind}_\mathcal{I}$ such that for every $i \in \text{Ind}_t$ we have $\mathcal{V}_t(i)\sigma_t = \mathcal{V}_\mathcal{I}(\psi(i))\sigma_\mathcal{I}$. *Wlog* we assume that ψ is defined as an extension of the connection between $\mathcal{C}_\mathcal{I}$ and $\mathcal{C}_t$, thereby ensuring that for input states i of $\mathcal{C}_t$ we also have $\mathcal{V}_t(i)\sigma'_t = \mathcal{V}_\mathcal{I}(\psi(i))\sigma'_\mathcal{I}$.

Claim. *Wlog we can assume that for any deduction state $i \in \text{IND}_t$ we have $\mathcal{V}_t(i)\sigma'_t \neq \mathcal{V}_\mathcal{I}(\psi(i))\sigma'_\mathcal{I}$.*

PROOF OF THE CLAIM. Let $i \in \text{IND}_t$ be a deduction state such that $\mathcal{V}_t(i)\sigma'_t = \mathcal{V}_\mathcal{I}(\psi(i))\sigma'_\mathcal{I}$. Adding a reuse state if necessary, we can change i into an input state that is connected to $\psi(t)$ (or a state which is a reuse of $\psi(i)$). This construction does not change σ_t nor σ'_t and thus the fact that $\mathcal{C}_t \circ \mathcal{C}_\mathcal{I} \circ \mathcal{C}_h$ or $\mathcal{C}_t \circ \mathcal{C}_\mathcal{I} \circ \mathcal{C}'_h$ is satisfiable. When repeatedly applying it, we obtain a symbolic derivation $\mathcal{C}_t$ that satisfies the claim. ◇

We now split the analysis in two cases depending on whether the set $I_t \subseteq \text{IND}_t$ of indices i such that $\mathcal{V}_t(i)\sigma'_t \neq \mathcal{V}_\mathcal{I}(\psi(i))\sigma'_\mathcal{I}$ is empty or not. If it is empty, the claim implies that we can assume there is no deduction states in $\mathcal{C}_t$, and thus that $\mathcal{S}_t = \mathcal{S}^t_t$. Since $\mathcal{C}_t \circ \mathcal{C}_\mathcal{I} \circ \mathcal{C}_h$ is satisfiable but not $\mathcal{C}_t \circ \mathcal{C}_\mathcal{I} \circ \mathcal{C}'_h$ there exists two input states i, j and one equation $\mathcal{V}_t(i) \stackrel{?}{=} \mathcal{V}_t(j)$ in $\mathcal{S}_t$ which is satisfied by σ_t but not by σ'_t. Thus $\chi(\mathcal{C}_\mathcal{I})$ contains one symbolic derivation $(\mathcal{V} : i \in \{1,2\} \mapsto x_i, \left\{x_1 \stackrel{?}{=} x_2\right\}, \emptyset, \{1,2\}, \emptyset)$ where 1 is connected to $\psi(i)$ and 2 is connected to $\psi(j)$.

On the other hand, if I_t is not empty, let i_0 be minimal in this set, and let $\mathcal{V}_t(i_0) \stackrel{?}{=} f(\mathcal{V}_t(i_1), \ldots, \mathcal{V}_t(i_n))$ be the equation corresponding to this deduction state in $\mathcal{S}^d_t$. Given the claim we can assume that i_0 is the first deduction state, and thus that all preceding states are input states. Thus there exists an ordering on the set $\text{IND}_0 = \{t, 0, \ldots, n\}$ such that the following symbolic derivation is in $\chi(\mathcal{C}_\mathcal{I})$ and satisfies the proposition:

$$(\mathcal{V} : i \in \text{IND}_0 \mapsto x_i, \left\{x_0 \stackrel{?}{=} f(x_1, \ldots, x_n)\ ,\ x_0 \stackrel{?}{=} x_t\right\}, \{t, 1, \ldots, n\}, \emptyset)$$

□

Given an HSD $\mathcal{C}_h$ let $\mathcal{C}^{\text{sf+}}_h \subseteq \mathcal{C}^\star_h$ be the set of ASDs which are the connection between an ASD in $\mathcal{C}^{\text{sf}}_h$ and a testing ASD as in Proposition 4. We again note that by construction the ASDs in $\mathcal{C}^{\text{sf+}}_h$ are aware.

Proposition 5 *Given two HSDs $\mathcal{C}_h$ and $\mathcal{C}'_h$ we have $\mathcal{C}^\star_h \subseteq \mathcal{C}'^\star_h$ if, and only if, $\mathcal{C}^{\text{sf+}}_h \subseteq \mathcal{C}'^\star_h$.*

PROOF. Since $\mathcal{C}^{\text{sf+}}_h \subseteq \mathcal{C}^\star_h$ the direct implication is trivial. Let us prove the converse direction by a contrapositive reasoning. If $\mathcal{C}^\star_h \not\subseteq \mathcal{C}'^\star_h$ then two cases are possible:

- If $\mathcal{C}^{\text{sf}}_h \not\subseteq \mathcal{C}'^\star_h$ then adding an extra deduction and a test equality $x \stackrel{?}{=} x$ to an ASD in $\mathcal{C}^{\text{sf}}_h \setminus \mathcal{C}'^\star_h$ yields an ASD in $\mathcal{C}^{\text{sf+}}_h \setminus \mathcal{C}'^\star_h$.

- Otherwise let $\mathcal{C}_\mathcal{I}$ be in $\mathcal{C}_h^\star \setminus \mathcal{C}_h'^\star$. By Lemma 7 there exists $\mathcal{C}'$ and $\mathcal{C}_t$ such that $\mathcal{C}' \circ \mathcal{C}_t \in \mathcal{C}_h^\star \setminus \mathcal{C}_h'^\star$. Since it contains $\mathcal{C}_t$ we have $\chi(\mathcal{C}') \neq \emptyset$. Thus by Proposition 4 it contains an ASD with at most one deduction state and one equality test $\mathcal{C}_t'$. We have $\mathcal{C}' \circ \mathcal{C}_t' \in \mathcal{C}_h^\star \setminus \mathcal{C}_h'^\star$ by definition, and thus $\mathcal{C}_h^{\text{sf+}} \not\subseteq \mathcal{C}_h' \star$

□

5.4.3 Reduction of $\mathcal{C}_h^{\text{sf+}} \subseteq \mathcal{C}_h'^\star$ to $\text{Sol}(\mathcal{C}_h) \subseteq \mathcal{C}_h'^\star$

We define in this subsection a set $\text{Sol}(\mathcal{C}_h)$ of ASDs for which it will be easy to provide a finite generating set. To this end we first prove a bound on the number of possible decompositions in stutter free ASDs. The construction then proceeds as follows. Given a HSD $\mathcal{C}_h$ we guess a number of deductions sufficient to include in particular the decompositions and the added deduction of Proposition 4, and a test between two terms (which may be or not in the guessed deductions). This is the set $\mathcal{D}ec(\mathcal{C}_h)$. We then consider all possible connections between an ASD $D \in \mathcal{D}ec(\mathcal{C}_h)$ and $\mathcal{C}_h$. For each resulting connection $\mathcal{C}$ we consider all stutter free and composition-only ASDs in $\mathcal{C}^\star$. The set $\text{Sol}(\mathcal{C}_h)$ is the set of connections between D and a composition-only stutter free ASD. We prove in Proposition 6 that $\text{Sol}(\mathcal{C}_h)$ contains in particular $\mathcal{C}_h^{\text{sf+}}$ thereby reducing our original goal of testing whether $\mathcal{C}_h^\star \subseteq \mathcal{C}_h'^\star$ to the simpler one $\text{Sol}(\mathcal{C}_h) \subseteq \mathcal{C}_h'^\star$.

Lemma 9 *Let $\mathcal{C}$ be a closed derivation, $\sigma = \text{Tr}_\mathcal{C}(\mathcal{C})$, and $t \in \text{Sub}(\sigma)$. Let $i_t \in \text{IND}$ be a minimal state such that $t \in \text{Sub}(\mathcal{V}(i_t)\sigma)$. Then either i_t is a knowledge state or a composition state.*

Proof. By minimality i_t cannot be a re-use state nor, by definition of decomposition rules and the fact that we consider a subterm deduction system, a decomposition state. Since the connection is closed it cannot be a reception state. □

In particular, let $\mathcal{C}_h = (\mathcal{V}_h, \mathcal{S}_h, \mathcal{K}_h, \text{IN}_h, \text{OUT}_h)$ be a HSD and consider an ASD $\mathcal{C}_\mathcal{I} = (\mathcal{V}_\mathcal{I}, \mathcal{S}_\mathcal{I}, \mathcal{K}_\mathcal{I}, \text{IN}_\mathcal{I}, \text{OUT}_\mathcal{I})$ in $\mathcal{C}_h^\star$. Let $\tau = \text{Tr}_{\mathcal{C}_h \circ \mathcal{C}_\mathcal{I}}(\mathcal{C}_h)$ and $\sigma = \text{Tr}_{\mathcal{C}_h \circ \mathcal{C}_\mathcal{I}}(\mathcal{C}_\mathcal{I})$. We say that a subterm t of $\mathcal{V}_\mathcal{I}(\text{IND}_\mathcal{I})\sigma$ is:

- *h-bound* if any minimal index $i \in \text{IND}_\mathcal{I}$ such that $t \in \text{Sub}(\mathcal{V}_\mathcal{I}(i)\sigma)$ is an input state (of $\mathcal{C}_\mathcal{I}$);
- *bound* if there exists a non-variable subterm s of $\mathcal{C}_h$ such that $(s\tau)\downarrow = t$;
- *free* otherwise.

Intuitively, h-bound terms are the terms built in the HSD, and free terms are those that can be replaced by any other term without changing the satisfiability of $\mathcal{S}_h$, and bound terms are built by the attacker but may be necessary for the satisfiability $\mathcal{S}_h$. We give in the next lemma a few properties of free and h-bound terms related to the deductions of the attacker. Note finally that the definitions above imply that a constant in $\mathrm{C}_{\mathrm{new}}$ is free.

Lemma 10 *Let $\mathcal{C}_h = (\mathcal{V}_h, \mathcal{S}_h, \mathcal{K}_h, \mathrm{In}_h, \mathrm{Out}_h)$ be a HSD, $\mathcal{C}_\mathcal{I}$ be in $\mathcal{C}_h^{\mathrm{sf}}$ with $\mathcal{C}_\mathcal{I} = (\mathcal{V}_\mathcal{I}, \mathcal{S}_\mathcal{I}, \mathcal{K}_\mathcal{I}, \mathrm{In}_\mathcal{I}, \mathrm{Out}_\mathcal{I})$, and $\sigma_x = \mathrm{Tr}_{\mathcal{C}_h \circ \mathcal{C}_\mathcal{I}}(\mathcal{C}_x)$ for $x \in \{h, \mathcal{I}\}$. For any term t in $\mathrm{Sub}(\sigma_\mathcal{I})$ we have:*

(i) If t is h-bound and $t \notin \mathrm{Sub}(\mathcal{K}_h \cup \mathcal{K}_\mathcal{I})$ there is a composition state $i' \in \mathrm{Ind}_h$ with $\mathcal{V}_h(i')\sigma_h = t$;

(ii) If t is h-bound then it is bound;

(iii) If $i \in \mathrm{Ind}_\mathcal{I}$ is a decomposition state the term $\mathcal{V}_\mathcal{I}(i)\sigma_\mathcal{I}$ is h-bound.

Proof. (i) Let $\mathcal{C} = (\mathcal{V}, \mathcal{S}, \mathcal{K}, \mathrm{In}, \mathrm{Out})$ be a closed and satisfiable composition of $\mathcal{C}_\mathcal{I}$ and $\mathcal{C}_h$. By Lemma 9 the minimal index $i \in \mathrm{Ind}$ such that $t \in \mathrm{Sub}(\mathcal{V}(i)\sigma_\mathcal{I})$ is either a knowledge or a composition state. Since $\mathcal{K} = \mathcal{K}_h \cup \mathcal{K}_\mathcal{I}$ and $t \notin \mathrm{Sub}(\mathcal{K}_h \cup \mathcal{K}_\mathcal{I})$ it must be a composition state. Since t is h-bound no minimal state in $\mathrm{Ind}_\mathcal{I}$ in which t appears is a deduction state. Since the connection preserves the ordering on $\mathrm{Ind}_\mathcal{I}$ the state i must be a composition state in Ind_h.

(ii) By the point (i) a h-bound term is either in $\mathcal{K}_h \cup \mathcal{K}_\mathcal{I}$ or is composed in some state i in $\mathcal{C}_h$. In the first case we note that $\mathrm{C}_{\mathrm{new}} \cap \mathrm{Sub}(\mathcal{C}_h) = \emptyset$ thus t h-bound implies that $t \in \mathcal{K}_h$ and hence is bound. In the second case and by definition of deduction states there exists in $\mathcal{S}_h$ an equation $\mathcal{V}_h(i) \stackrel{?}{=} f(\mathcal{V}_h(i_1), \ldots, \mathcal{V}_h(i_n))$ satisfied by σ_h. Since the state i is a composition state we thus have $\mathcal{V}_h(i)\sigma_h = f(\mathcal{V}_h(i_1)\sigma_h, \ldots, \mathcal{V}_h(i_n)\sigma_h) = t$. Hence t is bound.

(iii) Let i be a decomposition state and let $i_0 \leq i$ be a minimal index such that $\mathcal{V}_\mathcal{I}(i)\sigma_\mathcal{I} \in \mathrm{Sub}(\mathcal{V}_\mathcal{I}(i_0)\sigma_\mathcal{I})$. Since the equational theory is subterm the index i_0 cannot be a decomposition state by minimality of i_0 (and thus $i_0 \neq i$). If i_0 were a knowledge state, there would exist a constant $c \in \mathrm{C}_{\mathrm{new}}$ such that $\mathcal{V}_\mathcal{I}(i)\sigma_\mathcal{I} \in \mathrm{Sub}(V_\mathcal{I}(i_0)\sigma_\mathcal{I}) = \mathrm{Sub}(c)$ and thus we would have $\mathcal{V}_\mathcal{I}(i)\sigma_\mathcal{I} = c = V_\mathcal{I}(i_0)\sigma_\mathcal{I}$, which would contradict the fact that the derivation is stutter free. By minimality of i_0 it cannot be a re-use state. Finally if i_0 is a composition state we have in $\mathcal{S}_\mathcal{I}$ an equation $V_\mathcal{I}(i_0) \stackrel{?}{=} f(\mathcal{V}_\mathcal{I}(\alpha_1), \ldots, \mathcal{V}_\mathcal{I}(\alpha_k))$ which is satisfied by $\sigma_\mathcal{I}$

in the empty theory. Thus $\mathcal{V}_{\mathcal{I}}(i)\sigma_{\mathcal{I}} \in \mathrm{Sub}(V_{\mathcal{I}}(i_0)\sigma_{\mathcal{I}})$ implies either $\mathcal{V}_{\mathcal{I}}(i)\sigma_{\mathcal{I}} = V_{\mathcal{I}}(i_0)\sigma_{\mathcal{I}}$, which contradicts $\mathcal{C}_{\mathcal{I}}$ stutter free, or $\mathcal{V}_{\mathcal{I}}(i)\sigma_{\mathcal{I}} \in \mathrm{Sub}(\mathcal{V}_{\mathcal{I}}(\alpha_1)\sigma_{\mathcal{I}}, \ldots, \mathcal{V}_{\mathcal{I}}(\alpha_k)\sigma_{\mathcal{I}})$, which contradicts the minimality of i_0. Thus i_0 must be an input state. Thus all minimal indexes i_0 such that $\mathcal{V}_{\mathcal{I}}(i)\sigma_{\mathcal{I}} \in \mathrm{Sub}(\mathcal{V}_{\mathcal{I}}(i_0)\sigma_{\mathcal{I}})$ are input states, and therefore $\mathcal{V}_{\mathcal{I}}(i)\sigma_{\mathcal{I}}$ is h-bound by definition.

□

Lemma 10 details the structure of ASDs. Indeed, since decomposition can only be applied to deduce an h-bound term, and since h-bound terms are bound, we conclude from Lemma 10 that there are at most $|\mathrm{Sub}(\mathcal{C}_h)|$ decomposition states in a stutter free symbolic derivation of $\mathcal{C}_h^\star$.

The rest of the proof consists in bounding the number of *useful* compositions. We denote $\mathcal{C}_h^{\mathrm{comp}}$ the subset of $\mathcal{C}_h^{\mathrm{sf}}$ of ASD in which all deduction states are composition states. Given a HSD $\mathcal{C}_h = (\mathcal{V}_h, \mathcal{S}_h, \mathcal{K}_h, \mathrm{IN}_h, \mathrm{OUT}_h)$ and k the maximal arity of a public symbol we introduce two sets:

- $\mathcal{D}ec(\mathcal{C}_h)$ is the set of ASDs $\mathcal{C}$ with only reception, reuse, and deduction states, and less than $|\mathrm{Sub}(\mathcal{C}_h) + 1|$ deduction states, less than $|\mathrm{IN}_h|$ reuse states, less than $|\mathrm{OUT}_h| + k \cdot (|\mathrm{Sub}(\mathcal{C}_h)| + 1) + 1$ input states, and exactly one test equation.
- $\mathrm{Sol}(\mathcal{C}_h)$ is the set
$$\bigcup_{D \in \mathcal{D}ec(\mathcal{C}_h)} \bigcup_{\mathcal{C} \in (D \circ \mathcal{C}_h)^{\mathrm{comp}}} \mathcal{C} \circ D$$
where the union is over all possible symbolic derivations and all connections.

We note that by definition and since there is a finite number of public symbols and equations, the set $\mathcal{D}ec(\mathcal{C}_h)$ is finite.

Proposition 6 *$\mathcal{C}_h^{\mathrm{sf}+} \subseteq \mathrm{Sol}(\mathcal{C}_h)$ holds for any HSD $\mathcal{C}_h$.*

PROOF. Let $\mathcal{C}_h$ be a HSD, and let $\mathcal{C}_{\mathcal{I}}$ be in $\mathcal{C}_h^{\mathrm{sf}+}$. By definition of $\mathcal{C}_h^{\mathrm{sf}+}$ there exists $\mathcal{C}' \in \mathcal{C}_h^{\mathrm{sf}}$ and $\mathcal{C}_t$ such that $\mathcal{C}' \circ \mathcal{C}_t = \mathcal{C}_{\mathcal{I}}$. Since $\mathcal{C}_{\mathcal{I}}$ is in $\mathcal{C}_h^{\mathrm{sf}+}$ the ASD $\mathcal{C}_t$ has at most one deduction and one equality test.

By Proposition 3 there exists two ASDs $\mathcal{C}_c$ and $\mathcal{C}_d$ such that $\mathcal{C}_{\mathcal{I}} = \mathcal{C}_c \circ \mathcal{C}_d$ and such that in the connection $\mathcal{C}_h \circ \mathcal{C}_c \circ \mathcal{C}_d$ the deduction states of $\mathcal{C}_c$ (*resp.* $\mathcal{C}_d$) are composition states (*resp.* decomposition states), and let $\mathcal{C}'_h = \mathcal{C}_d \circ \mathcal{C}_h = (\mathcal{V}'_h, \mathcal{S}'_h, \mathcal{K}'_h, \mathrm{IN}'_h, \mathrm{OUT}'_h)$. Let us note $d(\mathcal{C}_{\mathcal{I}})$ the derivation $\mathcal{C}_d$. Using these notations, let:
$$\mathcal{D} = \bigcup_{\mathcal{C}_{\mathcal{I}} \in \mathcal{C}_h^{\mathrm{sf}+}} d(\mathcal{C}_{\mathcal{I}})$$

By Lemma 10, point iii for every $\mathcal{C}_\mathcal{I} \in \mathcal{C}_h^{\text{sf+}}$ the derivation $d(\mathcal{C}_\mathcal{I})$ has at most $|\text{Sub}(\mathcal{C}_h)|$ decomposition states that are present in $\mathcal{C}'$, and at most one decomposition state in $\mathcal{C}_t$. Furthermore it has at most one equality test (none in $\mathcal{C}'$, one in $\mathcal{C}_t$). Thus it is in $\mathcal{D}ec(\mathcal{C}_h)$ and we have $\mathcal{D} \subseteq \mathcal{D}ec(\mathcal{C}_h)$. Since $\mathcal{C}_\mathcal{I}$ is in $\mathcal{C}_h^{\text{sf+}}$, we additionally know that $\mathcal{C}_c$ is a stutter free derivation in the connection $\mathcal{C}'_h \circ \mathcal{C}_c$. Since all its deduction states are compositions, we have that $\mathcal{C}_c \in (\mathcal{C}_h \circ \mathcal{C}_d)^{\text{comp}}$. □

5.4.4 Reduction of $\text{Sol}(\mathcal{C}_h) \subseteq \mathcal{C}_h'^\star$ to $\min_<(\text{Sol}(\mathcal{C}_h)) \subseteq \mathcal{C}_h'^\star$

The ASDs in $\text{Sol}(\mathcal{C}_h)$ have the property that, when replacing a constant in C_{new} by the result of a sequence of compositions (this operation is called *opening*) we obtain another ASD which can be connected to all the HSDs the original ASD could be connected to (Lemma 11). Defining minimal ASDs in $\text{Sol}(\mathcal{C}_h)$ to be the ones which, by this opening operation, generates all ASDs in $\text{Sol}(\mathcal{C}_h)$ it is then trivial to check the inclusion $\mathcal{C}_h^{\text{sf+}} \subseteq \mathcal{C}_h'^\star$: it suffices to check whether $\min_<(\text{Sol}(\mathcal{C}_h)) \subseteq \mathcal{C}_h'^\star$ (Lemma 12).

Opening of symbolic derivations. If $\mathcal{C} = (\mathcal{V}, \mathcal{S}, \mathcal{K}, \text{In}, \text{Out})$ and C is a subset of constants in $\mathcal{K}$, we *open* $\mathcal{C}$ *on* C, and denote the operation $\text{open}_C(\mathcal{C})$ when, for each $c \in C$:

- If $i \in \text{Ind}$ is the first knowledge state with $\mathcal{V}(i) \stackrel{?}{=} c \in \mathcal{S}$, we remove this equation from $\mathcal{S}$ and add i to the input states;
- we replace all occurrences of c in $\mathcal{C}$ by $\mathcal{V}(i)$.

Lemma 11 *Let $\mathcal{C}_\mathcal{I} \in \mathcal{C}_h^\star$ with $\mathcal{C}_\mathcal{I} = (\mathcal{V}_\mathcal{I}, \mathcal{S}_\mathcal{I}, \mathcal{K}_\mathcal{I}, \text{In}_\mathcal{I}, \text{Out}_\mathcal{I})$, let $C \subseteq \mathcal{K}_\mathcal{I}$ and let $\mathcal{C}_c \in \mathcal{C}_h'^{\text{sf}}$ for some HSD $\mathcal{C}'_h$. If a connection $\mathcal{C}_c \circ \mathcal{C}_h \circ \text{open}_C(\mathcal{C}_\mathcal{I})$ is closed then it is satisfiable.*

Proof. Let us first introduce some notations. We denote:

$$\begin{cases} \mathcal{C}_h &= (\mathcal{V}_h, \mathcal{S}_h, \mathcal{K}_h, \text{In}_h, \text{Out}_h) \\ \mathcal{C}_c &= (\mathcal{V}_c, \mathcal{S}_c, \mathcal{K}_c, \text{In}_c, \text{Out}_c) \\ \mathcal{C}'_\mathcal{I} &= \mathcal{C}_c \circ \text{open}_C(\mathcal{C}_\mathcal{I}) \\ \mathcal{C}'_\mathcal{I} &= (\mathcal{V}'_\mathcal{I}, \mathcal{S}'_\mathcal{I}, \mathcal{K}'_\mathcal{I}, \text{In}'_\mathcal{I}, \text{Out}'_\mathcal{I}) \end{cases}$$

By Proposition 2, point *ii*, the substitution $\text{Tr}_{\mathcal{C}_c \circ \mathcal{C}_h \circ \text{open}_C(\mathcal{C}_\mathcal{I})}(\mathcal{C}_c)$ satisfies $\mathcal{S}_c$. Since $\mathcal{C}_c$ is an ASD we have $C \subset \text{C}_{\text{new}}$. Since $\mathcal{C}_h$ is a HSD every constant in C is distinct from the constants appearing in $\mathcal{S}_h$. Lemma 4 and $\sigma \models \mathcal{S}_h \circ \mathcal{S}_\mathcal{I}$ then imply $\sigma\delta_{c,t} \models \mathcal{S}_h$. By definition of ASDs no constant in C appears in $\mathcal{S}_\mathcal{I}$ once the corresponding knowledge equations are removed by the opening.

Let $\sigma' = \mathrm{Tr}_{\mathcal{C}_c \circ \mathcal{C}_h \circ \mathrm{open}_C(\mathcal{C}_\mathcal{I})}(\mathcal{C}_\mathcal{I})$. For each memory state $i \in \textsc{Ind}_\mathcal{I}$ that contains a constant $c \in C$ we let $t_c = \mathcal{V}_\mathcal{I}(i)\sigma'$. We define δ as the replacement of each constant $c \in C$ by the term t_c.

By induction on the indexes of the connection $\mathcal{C}_c \circ \mathcal{C}_h \circ \mathrm{open}_C(\mathcal{C}_\mathcal{I})$ we have:

$$\mathrm{Tr}_{\mathcal{C}_c \circ \mathcal{C}_h \circ \mathrm{open}_C(\mathcal{C}_\mathcal{I})}(\mathcal{C}_c \circ \mathcal{C}_h \circ \mathrm{open}_C(\mathcal{C}_\mathcal{I})) = \mathrm{Tr}_{\mathcal{C}_h \circ \mathcal{C}_\mathcal{I}}(\mathcal{C}_h \circ \mathcal{C}_\mathcal{I})\delta$$

Thus every equation in $\mathcal{S}_h \cup \mathcal{S}_\mathcal{I}$ (minus the removed memory equations) is satisfied by the composition with $\mathcal{C}_c$. Since every equation in its unification system is satisfied the connection $\mathcal{C}_c \circ \mathcal{C}_h \circ \mathrm{open}_C(\mathcal{C}_\mathcal{I})$ is satisfiable. □

Ordering on symbolic derivations. Given two symbolic derivations $\mathcal{C}_\mathcal{I} = (\mathcal{V}_\mathcal{I}, \mathcal{S}_\mathcal{I}, \mathcal{K}_\mathcal{I}, \textsc{In}_\mathcal{I}, \textsc{Out}_\mathcal{I})$ and $\mathcal{C}'_\mathcal{I} = (\mathcal{V}'_\mathcal{I}, \mathcal{S}'_\mathcal{I}, \mathcal{K}'_\mathcal{I}, \textsc{In}'_\mathcal{I}, \textsc{Out}'_\mathcal{I})$, we say that $\mathcal{C}_\mathcal{I} < \mathcal{C}'_\mathcal{I}$ if:

- there exists $C \subseteq \mathcal{K}_\mathcal{I}$, a stutter free symbolic derivation $\mathcal{C}_C$ and a connection φ such that $\mathcal{C}_C \circ_\varphi \mathrm{open}_C(\mathcal{C}_\mathcal{I}) = \mathcal{C}'_\mathcal{I}$ modulo a renaming of variables;
- or there exists a set of memory states $I \subseteq \textsc{Ind}'_\mathcal{I}$ such that $\mathcal{C}_\mathcal{I}$ is equal to $\mathcal{C}''_\mathcal{I} = (\mathcal{V}''_\mathcal{I}, \mathcal{S}''_\mathcal{I}, \mathcal{K}''_\mathcal{I}, \textsc{In}''_\mathcal{I}, \textsc{Out}''_\mathcal{I})$ where:
  - $\mathcal{V}''_\mathcal{I}$ is the restriction of $\mathcal{V}'_\mathcal{I}$ to the domain $\textsc{Ind}'_\mathcal{I} \setminus I$ and $\mathcal{S}''_\mathcal{I} = \mathcal{S}'_\mathcal{I} \setminus \left\{ \mathcal{V}'_\mathcal{I}(i) \stackrel{?}{=} c_i \right\}_{i \in I}$.

Since $\mathcal{C}''_\mathcal{I}$ is a symbolic derivation, we note that the memory states of $\mathcal{C}'_\mathcal{I}$ that are removed are never re-used nor employed in any deduction. Given that $\mathcal{C} < \mathcal{C}'$ implies that $\mathcal{C}$ has less indexes than $\mathcal{C}'$, it is clear that the relation $<$ is a well-founded ordering relation.

Lemma 12 *Let $\mathcal{C}_h$ and $\mathcal{C}'_h$ be two HSDs. If $\min_<(\mathrm{Sol}(\mathcal{C}_h)) \subseteq \mathcal{C}'^\star_h$ then $\mathrm{Sol}(\mathcal{C}_h) \subseteq \mathcal{C}'^\star_h$.*

Proof. Assume $\min_<(\mathrm{Sol}(\mathcal{C}_h)) \subseteq \mathcal{C}'^\star_h$ and let $\mathcal{C}_\mathcal{I}$ be in $\mathrm{Sol}(\mathcal{C}_h)$. By definition of the ordering there exists a derivation $\mathcal{C}'_\mathcal{I} \in \min_<(\mathrm{Sol}(\mathcal{C}_h))$, a set of constants C and a composition-only derivation $\mathcal{C}_c$ such that $\mathcal{C}_c \circ \mathrm{open}_C(\mathcal{C}'_\mathcal{I}) = \mathcal{C}_\mathcal{I}$. By hypothesis we have $\mathcal{C}'_\mathcal{I} \in \mathcal{C}'^\star_h$. By Lemma 11 this implies that $\mathcal{C}_\mathcal{I}$ is also in $\mathcal{C}'^\star_h$. □

5.4.5 Decision procedure for $\min_<(\mathrm{Sol}(\mathcal{C}_h)) \subseteq \mathcal{C}'^\star_h$

We conclude by giving a bound on the size of ASDs in $\min_<(\mathrm{Sol}(\mathcal{C}_h))$ (Proposition 7). This bound implies the completeness of the Inclusion$(\mathcal{C}_h, \mathcal{C}'_h)$ algorithm. Since this algorithm rejects the inclusion when it finds an ASD in $\mathcal{C}^\star_h \setminus \mathcal{C}'^\star_h$ its soundness is trivial (Theorem 2). Our main theorem is then a direct consequence.

Proposition 7 *There exists a polynomial P such that for any HSD $\mathcal{C}_h$ every $\mathcal{C}_\mathcal{I} \in \min_<(\mathrm{Sol}(\mathcal{C}_h))$ is of size smaller than $P(|\mathrm{Sub}(\mathcal{C}_h)|)$.*

Proof. Let $\mathcal{C}_\mathcal{I}$ be a symbolic derivation in $\min_<(\mathrm{Sol}(\mathcal{C}_h))$. By definition of $\mathrm{Sol}(\mathcal{C}_h)$ there exists an ASD $\mathcal{C}_d$ with less than $|\mathrm{Sub}(\mathcal{C}_h)|+1$ deduction and a composition-only symbolic derivation $\mathcal{C}_c$ such that $\mathcal{C}_\mathcal{I} = \mathcal{C}_d \circ \mathcal{C}_c$. Consider the HSD $\mathcal{C}'_h = \mathcal{C}_h \circ \mathcal{C}_d$ and let $\mathcal{C}'_h = (\mathcal{V}'_h, \mathcal{S}'_h, \mathcal{K}'_h, \mathrm{IN}'_h, \mathrm{OUT}'_h)$ and $\sigma = \mathrm{Tr}_{\mathcal{C}'_h \circ \mathcal{C}_c}(\mathcal{C}'_h \circ \mathcal{C}_c)$.

We have seen that basic narrowing provides a complete unification procedure for subterm convergent equational theories, and thus we can assume that after a sequence of narrowing steps of length bounded by $|\mathrm{Sub}(\mathcal{S}_h)|$, for every $t \in \mathrm{Sub}(\mathcal{S}'_h)$ we have $t\sigma = (t\sigma)\!\downarrow$ (applying narrowing steps) and the number of different terms in $\mathcal{C}'_h$ is linear *w.r.t* the number of terms in $\mathcal{C}_h$. In the rest of the proof the notion of free term in $\mathcal{C}_c$ is *w.r.t.* the connection with $\mathcal{C}'_h$ and considered after the narrowing steps are applied.

Claim. *If there exists a deduction state i in $\mathcal{C}_c$ such that $\mathcal{V}_c(i)\sigma$ is free then $\mathcal{C}_c$ is not minimal.*

Proof of the claim. Assume that there exists a deduction state i in $\mathcal{C}_c$ such that $t = \mathcal{V}_c(i)\sigma$ is free and let c be a new constant. Let us consider the symbolic derivation $\mathcal{C}'_\mathcal{I}$ equal to $\mathcal{C}_\mathcal{I}$ but on state i which becomes a memory state with an equation $\mathcal{V}'_\mathcal{I}(i) \stackrel{?}{=} c$. Noting that after guessing the narrowing steps the unification system $\mathcal{S}'_h$ is satisfied by σ *in the empty theory* we have:

- Since t is free, for any term $s \in \mathrm{Sub}(\mathcal{S}'_h)$ we have $(s\sigma)\delta_{t,c} = s(\sigma\delta_{t,c})$ by Lemma 5
- By induction on the states using the first point, $\mathrm{Tr}_{\mathcal{C}'_\mathcal{I} \circ \mathcal{C}'_h}(\mathcal{C}'_\mathcal{I} \circ \mathcal{C}'_h) = \sigma\delta_{t,c}$;
- As a consequence, $\mathcal{C}'_\mathcal{I}$ is also in $\mathcal{C}'^\star_h$.

◇

We are now able to conclude: since by definition $\mathcal{C}'_{\mathcal{I}} < \mathcal{C}_{\mathcal{I}}$ the symbolic derivation $\mathcal{C}_{\mathcal{I}}$ is not minimal. Thus by contrapositive reasoning, all minimal solutions are such that the result of a composition is bounded. They have thus less than $|\mathrm{Sub}(\mathcal{C}'_h)|$ composition steps. In the worst case this implies that there is $k \cdot |\mathrm{Sub}(\mathcal{C}'_h)|$ memory or re-use states. Since the connection with $\mathcal{C}'_h$ must be closed, we can bound the number of reception states by the number of output state of $\mathcal{C}'_h$. □

From the proof we note that the polynomial P depends on the equational theory and on the public symbols.

Theorem 2 *(Inclusion of $\mathcal{C}^\star_h$ into $\mathcal{C}'^\star_h$) If Algorithm* Inclusion$(\mathcal{C}_h, \mathcal{C}'_h)$ *always returns* SAT *then* $\mathcal{C}^\star_h \subseteq \mathcal{C}'^\star_h$.

PROOF. On the one hand, we note that in particular the algorithm guesses all the symbolic derivations in $\min_<(\mathrm{Sol}(\mathcal{C}_h))$. Thus, if it always returns SAT we have $\mathrm{Sol}(\mathcal{C}_h) \subseteq \mathcal{C}'^\star_h$ by Lemma 12. Thus by Proposition 6 we have $\mathcal{C}^{\mathrm{sf}+}_h \subseteq \mathcal{C}'^\star_h$. Proposition 5 then implies $\mathcal{C}^\star_h \subseteq \mathcal{C}'^\star_h$. On the other hand, if the algorithm fails it has found one ASD in $\mathcal{C}^\star_h \setminus \mathcal{C}'^\star_h$ and thus the inclusion does not hold. □

Inclusion$(\mathcal{C}_h, \mathcal{C}'_h)$

Algorithm. Guess a symbolic derivation $\mathcal{C}_{\mathcal{I}}$ of size less than $P(\mathcal{C}_h)$ and a connection φ between $\mathcal{C}_{\mathcal{I}}$ and $\mathcal{C}_h$

Output: If $\mathcal{C}_{\mathcal{I}} \circ \mathcal{C}_h$ satisfiable implies $\mathcal{C}_{\mathcal{I}} \circ \mathcal{C}'_h$ satisfiable then SAT, else FAIL.

As a direct consequence we obtain the announced theorem.

Theorem 1. *Symbolic equivalence is decidable for subterm deduction systems.*

5.4.6 Discussion on complexity

The exact complexity of the inclusion algorithm depends on the assumptions made about the class of HSDs we consider as well as on the modeling of cryptographic primitives.

HSDs representing protocols We have studied in [32] the conditions under which a protocol narration (*i.e.*, a sequence of messages in the Alice&Bob notation and the specification of the initial knowledge of participants) can be turned into symbolic derivations of the type presented in Example 5.2. In particular, the compilation method described will succeed in producing a HSD for any subterm deduction system when the protocol narration is executable.

The HSDs produced have a property relevant to the complexity analysis of the equivalence problem. In this case the unification system contains only equations that are equalities between contexts over the input variables of the symbolic derivation. This property is always true for ASDs by definition.

In the Inclusion$(\mathcal{C}_h, \mathcal{C}'_h)$ algorithm, once the connection is guessed the symbolic derivations $\mathcal{C}_\mathcal{I} \circ \mathcal{C}_h$ and $\mathcal{C}_\mathcal{I} \circ \mathcal{C}'_h$ are closed. By Lemma 6 this implies that all input variables of the honest and attacker symbolic derivations are instantiated by ground terms, and thus the unification systems are actually sets of equalities between ground terms. These equalities can be checked in polynomial time.

More formally and when $\mathcal{I}$ is a subterm deduction system let us define COMPILED_EQ to be the set of instances of the $\mathcal{I}$-symbolic equivalence problem in which the input HSDs $\mathcal{C}_h = (\mathcal{V}_h, \mathcal{S}_h, \mathcal{K}_h, \text{IN}_h, \text{OUT}_h)$ are such that $\text{Var}(\mathcal{S}_h) \subseteq \text{Var}(\mathcal{V}_h(\text{IND}_h))$.

Corollary 1 COMPILED_EQ *is in coNP.*

We believe that COMPILED_EQ covers most of the equivalence problems encountered when analyzing cryptographic protocols, *i.e.*, those obtained by compilation of an Alice&Bob specification [32].

Example 5.7 *One may add to the HSD representing the role B in Example 5.2 an equation* $x_5 \stackrel{?}{=} \text{ae}(y, x_3)$ *to model that B can test that the received message is a proper encryption with his public key. This test introduces a variable which is not in the image of* $\mathcal{V}$ *and thus testing the equivalence of the modified HSD with another one is not in* COMPILED_EQ.

We however note that an alternative modeling would have been to add a test symbol to the signature. For instance if we mandate that one needs the public key to check whether a message is a proper encryption we would add the public symbols check_enc *(of arity 2) and* $\top$ *(of arity* 0*) with the equation:*

$$\text{check\_enc}(\text{ae}(x, y), y) = \top$$

With this model we add to the HSD modeling B in Example 5.2 an equation that can only be satisfied when the received message x_5 *is a proper encryption*

with public key $\mathrm{pk}(B)$*:*

$$\mathrm{check\_enc}(x_5, x_3) \stackrel{?}{=} \top$$

This example also demonstrates why we believe the class COMPILED_EQ is sufficient since the ultimate conclusion of [32] is that *when all possible operations are made explicit in the deduction system* every executable protocol narration is compiled into a HSD that satisfies the constraints of COMPILED_EQ.

Modeling issues In relation with the discussion in Section 5.3.3 one could choose to model non silently failing cryptographic operations. In this case the attacker can build more discriminating tests based on the success or failure of the operations. Hence we have to modify the decision problem accordingly as follows.

Let us furthermore assume that all rules in the rewriting system are of the form $f(t_1, \ldots, t_n) \rightarrow t$ with f a public symbol. In this case the fact that application of the function modeled by f succeeds or fails is represented symbolically by the success or failure of a rewrite step.

To solve the equivalence problem corresponding to this cryptographic hypothesis one has to:

- Change the $\mathcal{I}$-symbolic equivalence problem by additionally requiring that the composition and decomposition states of $\mathcal{C}_\mathcal{I}$ are the same in the two connections with $\mathcal{C}_h$ and $\mathcal{C}'_h$;
- Change the inclusion algorithm corresponding to this new equivalence problem by adding a step in which the decomposition states are guessed.

Once the guesses are performed we can assume no other rewrite rule is applied on a term in the ASD. As we have seen in Section 5.3.3 it makes sense in this case to also assume that the rewriting steps that are applied on the terms in the HSD are fixed. As a consequence, checking the satisfiability of a connection is reduced to checking the satisfiability of unification systems when no equation is applied, *i.e.*, in the empty theory. This problem is polynomial, and thus in this case the symbolic equivalence problem is again in $coNP$. In this case the HSDs do not have to be obtained by compilation of an Alice&Bob specification.

5.4.7 Extension to ground right-hand side

The definition of subterm equational theories in [12, 13] is slightly more general than the one we employed. In addition to rewrite rules $l \to r$ such that r is a subterm of l, a rewrite rule $l \to r$ may also be such that r is a ground term. In order to extend our results to this case we have to assume that every HSD contains in its memory states every ground right-hand side of the rewrite rules. As long as these added memory states are not in OUT, this change does not alter the set of solutions of a HSD. In return this construction permits us to extend the set of guessed terms to the subterms of the ground right-hand sides of rewrite rules. The definitions of stutter free derivations, composition and decomposition states are left as is. We note that the proof of Lemma 9 is not changed with this new definition of subterm deduction system. For Lemma 10, we note that point i assumes conditions that are not satisfied by the added terms (which are bound), and that having added the ground right-hand sides of the rewrite rules implies point iii. The proof of point ii is unchanged. With our definition of decomposition rules, the rest of the proofs do not change.

5.5 Concluding remarks

We believe that symbolic derivations offer an interesting framework to study and prove equivalence-based security properties. The decision procedure derived here is simple but cannot be implemented as such. Future work on this topic will be focused on the implementability of this decision procedure, and in particular on giving conditions on the deduction system such that the AVANTSSAR tools can be easily extended to solve equivalence problems.

6 Conclusions

Our results show that it is possible in many situations to model service behaviors and their components or aspects (policies, workflows, databases) as transitions systems, Horn clauses or constraints and provide effective algorithms to validate complex security properties on these models.

Horn clauses models for databases introduce new undesired service behaviors and therefore may generate false positives. Hence, in future works, we may investigate techniques for limiting their impact. Reachability problems that stem from service verification can also be encoded as constraints and existing modularity results about constraint satisfiability can be exploited in order to combine access control calculus and message-based transitions. More work is necessary in order to address more complex workflows and policies, and to prove equivalence properties defined through indistinguishability.

References

[1] M. Abadi, M. Baudet, and B. Warinschi. Guessing attacks and the computational soundness of static equivalence. In L. Aceto and A. Ingólfsdóttir, editors, *Foundations of Software Science and Computation Structures, 9th International Conference, FOSSACS 2006, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2006, Vienna, Austria, March 25-31, 2006, Proceedings*, volume 3921 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 2006.

[2] M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *Proceedings of the Principle of Programming Languages Conference*, pages 104–115, 2001.

[3] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. In *ACM Conference on Computer and Communications Security*, pages 36–47, 1997.

[4] M. Abadi and P. Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *J. Cryptol.*, 20(3):395–395, 2007.

[5] A. Armando and L. Compagna. SAT-based Model-Checking for Security Protocols Analysis. *International Journal of Information Security*, 7(1):3–32, 2008.

[6] N. Asokan, V. Shoup, and M. Waidner. Asynchronous protocols for optimistic fair exchange. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 86–99, 1998. citeseer.nj.nec.com/asokan98asynchronous.html.

[7] AVANTSSAR. Deliverable 3.3: Attacker models. Available at http://www.avantssar.eu, 2008.

[8] AVANTSSAR. Deliverable 2.3: ASLan final version with dynamic service and policy composition. Available at http://www.avantssar.eu, 2010.

[9] AVISPA. Deliverable 2.1: The High-Level Protocol Specification Language. http://www.avispa-project.org, 2003.

[10] D. Basin and H. Ganzinger. Automated complexity analysis based on ordered resolution. *J. ACM*, 48(1):70–109, 2001.

[11] D. Basin, S. Mödersheim, and L. Viganò. OFMC: A symbolic model checker for security protocols. *International Journal of Information Security*, 4(3):181–208, 2005.

[12] M. Baudet. Deciding security of protocols against off-line guessing attacks. In V. Atluri, C. Meadows, and A. Juels, editors, *ACM Conference on Computer and Communications Security*, pages 16–25. ACM, 2005.

[13] M. Baudet. *Sécurité des protocoles cryptographiques : aspects logiques et calculatoires*. Thèse de doctorat, Laboratoire Spécification et Vérification, ENS Cachan, France, Jan. 2007.

[14] M. Y. Becker, C. Fournet, and A. D. Gordon. Security Policy Assertion Language (SecPAL). http://research.microsoft.com/en-us/projects/SecPAL/.

[15] J. Bengtson, K. Bhargavan, C. Fournet, A. D. Gordon, and S. Maffeis. Refinement types for secure implementations. In *CSF*, pages 17–32, 2008.

[16] K. Bhargavan, C. Fournet, A. D. Gordon, and R. Pucella. Tulafale: A security tool for web services. In F. S. de Boer, M. M. Bonsangue, S. Graf, and W. P. de Roever, editors, *Formal Methods for Components and Objects, Second International Symposium, FMCO 2003, Leiden, The Netherlands, November 4-7, 2003, Revised Lectures*, volume 3188 of *Lecture Notes in Computer Science*, pages 197–222. Springer, 2003.

[17] B. Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *Proceedings of CSFW'01*, pages 82–96. IEEE Computer Society Press, 2001.

[18] B. Blanchet. Automatic proof of strong secrecy for security protocols. In *IEEE Symposium on Security and Privacy*, pages 86–. IEEE Computer Society, 2004.

[19] B. Blanchet. Automatic verification of correspondences for security protocols. *Journal of Computer Security*, 17(4):363–434, July 2009.

[20] B. Blanchet, M. Abadi, and C. Fournet. Automated verification of selected equivalences for security protocols. In *LICS*, pages 331–340. IEEE Computer Society, 2005.

[21] C. Bodei, M. Buchholtz, P. Degano, F. Nielson, and H. R. Nielson. Static validation of security protocols. *Journal of Computer Security*, 13(3):347–390, 2005.

[22] Y. Boichut, P.-C. Heam, O. Kouchnarenko, and F. Oehl. Improvements on the Genet and Klay Technique to Automatically Verify Security Protocols. In *Proc. Int. Workshop on Automated Verification of Infinite-State Systems (AVIS'2004), joint to ETAPS'04*, pages 1–11, Barcelona, Spain, 2004. The final version will be published in EN in Theoretical Computer Science, Elsevier.

[23] Y. Boichut, P.-C. Héam, O. Kouchnarenko, and F. Oehl. Improvements on the Genet and Klay technique to automatically verify security protocols. In *AVIS'04*, pages 1–11, 2004.

[24] M. Boreale, R. D. Nicola, and R. Pugliese. Proof techniques for cryptographic processes. In *LICS*, pages 157–166, 1999.

[25] L. Bozga, Y. Lakhnech, and M. Perin. Hermes: An automatic tool for the verification of secrecy in security protocols. In *CAV'03*, LNCS 2725, pages 219–222. Springer-Verlag, 2003.

[26] L. Bozga, Y. Lakhnech, and M. Perin. Pattern-based abstraction for verifying secrecy in protocols. In *Proceedings of TACAS 2003*, LNCS 2619. Springer-Verlag, 2003.

[27] P. Broadfoot, G. Lowe, and A. Roscoe. Automating data independence. In *Proceedings of Esorics 2000*, LNCS 1895, pages 175–190. Springer-Verlag, 2000.

[28] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings of FOCS 2001*, pages 136–145. IEEE Computer Society, 2001. Long version available as http://eprint.iacr.org/2000/067.ps.

[29] V. Cheval, H. Comon-Lundh, and S. Delaune. Automating security analysis: Symbolic equivalence of constraint systems. In *Automated Reasoning, 5th International Joint Conference, IJCAR 2010, Edinburgh, UK, July 16-19, 2010. Proceedings*, volume 6173 of *Lecture Notes in Computer Science*. Springer, 2010.

[30] Y. Chevalier and M. Kourjieh. On the decidability of (ground) reachability problems for cryptographic protocols (extended version). Technical report, INRIA, 2008. available at http://hal.inria.fr/inria-00392226_v1/.

[31] Y. Chevalier and M. Rusinowitch. Combining intruder theories. In L. Caires, G. F. Italiano, L. Monteiro, C. Palamidessi, and M. Yung,

editors, *ICALP*, volume 3580 of *Lecture Notes in Computer Science*, pages 639–651. Springer, 2005.

[32] Y. Chevalier and M. Rusinowitch. Compiling and securing cryptographic protocols. *Inf. Process. Lett.*, 110(3):116–122, 2010.

[33] Y. Chevalier and L. Vigneron. Automated Unbounded Verification of Security Protocols. In *Proceedings of CAV'2002*, LNCS 2404, pages 324–337. Springer, 2002. http://www.loria.fr/~vigneron/Work/papers/ChevalierV-CAV02.ps.gz.

[34] H. Comon-Lundh and V. Cortier. Computational soundness of observational equivalence. In *ACM Conference on Computer and Communications Security*, pages 109–118, 2008.

[35] V. Cortier and S. Delaune. A method for proving observational equivalence. In *Proceedings of the 22nd IEEE Computer Security Foundations Symposium (CSF'09)*, pages 266–276. IEEE Computer Society Press, 2009.

[36] S. Delaune, S. Kremer, and M. D. Ryan. Verifying privacy-type properties of electronic voting protocols: A taster. In D. Chaum, M. Jakobsson, R. L. Rivest, P. Y. A. Ryan, J. Benaloh, M. Kutyłowski, and B. Adida, editors, *Towards Trustworthy Elections – New Directions in Electronic Voting*, volume 6000 of *Lecture Notes in Computer Science*, pages 289–309. Springer, May 2010.

[37] J. DeTreville. Binder, a logic-based security language. In *IEEE Symposium on Security and Privacy*, pages 105–113, 2002.

[38] Y. Ding and P. Horster. Undetectable on-line password guessing attacks. *Operating Systems Review*, 29(4):77–86, 1995.

[39] D. Dolev and A. Yao. On the Security of Public-Key Protocols. *IEEE Transactions on Information Theory*, 2(29), 1983.

[40] N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. Undecidability of bounded security protocols. In *Proceedings of the Workshop on Formal Methods and Security Protocols*, 1999.

[41] S. Even and O. Goldreich. On the security of multi-party ping-pong protocols. In *FOCS*, pages 34–39, 1983.

[42] T. Genet and F. Klay. Rewriting for cryptographic protocol verification. In *Proceedings of CADE'00*, LNCS 1831, pages 271–290. Springer-Verlag, 2000.

[43] Y. Gurevich and I. Neeman. DKAL: Distributed-knowledge authorization language. In *Proceedings of CSF 2008*, pages 149–162. IEEE Computer Society, 2008.

[44] Y. Gurevich and I. Neeman. The logic of infons. *Bulletin of the EATCS*, (98):150–178, 2009.

[45] J. Heather, G. Lowe, and S. Schneider. How to prevent type flaw attacks on security protocols. In *Proceedings of The 13th Computer Security Foundations Workshop (CSFW'00)*. IEEE Computer Society Press, 2000.

[46] J.-M. Hullot. Canonical forms and unification. In W. Bibel and R. A. Kowalski, editors, *CADE*, volume 87 of *Lecture Notes in Computer Science*, pages 318–334. Springer, 1980.

[47] H. Hüttel. Deciding framed bisimilarity. Presented at the INFINITY'02 workshop, June 2002.

[48] R. Küsters and M. Tuengerthal. Universally Composable Symmetric Encryption. In *Proceedings of the 22nd IEEE Computer Security Foundations Symposium (CSF 2009)*, pages 293–307. IEEE Computer Society, 2009.

[49] G. Lowe. Breaking and Fixing the Needham-Shroeder Public-Key Protocol Using FDR. In T. Margaria and B. Steffen, editors, *Proceedings of TACAS'96*, LNCS 1055, pages 147–166. Springer-Verlag, 1996.

[50] G. Lowe. Casper: a Compiler for the Analysis of Security Protocols. *Journal of Computer Security*, 6(1):53–84, 1998. http://web.comlab.ox.ac.uk/oucl/work/gavin.lowe/Security/Casper/.

[51] J. K. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *Proceedings of the ACM Conference on Computer and Communications Security CCS'01*, pages 166–175, 2001.

[52] S. Mödersheim. On the Relationships between Models in Protocol Verification (extended version). Technical Report 512, ETH Zurich, Dep. of Computer Science, 2006. Updated version in Journal of Information and Computation, 206(2–4).

[53] S. Mödersheim. On the Relationships between Models in Protocol Verification. *Journal of Information and Computation*, 206(2–4):291–311, 2008. http://dx.doi.org/10.1016/j.ic.2007.07.006.

[54] S. Mödersheim. Verification based on set-abstraction using the AIF framework. Technical Report IMM-Technical report-2010-09, DTU/IMM, 2010. www.imm.dtu.dk/~samo.

[55] M. Rusinowitch and M. Turuani. Protocol insecurity with finite number of sessions is NP-complete. In *Proceedings of CSFW'01*, pages 174–190. IEEE Computer Society Press, 2001.

[56] M. Rusinowitch and M. Turuani. Protocol Insecurity with Finite Number of Sessions and Composed Keys is NP-complete. *Theoretical Computer Science*, 299:451–475, 2003. http://www.avispa-project.org.

[57] B. Schneier. *Applied cryptography*. Addison-Wesley, 1996.

[58] SEVECOM. Deliverable 2.1-App.A: Baseline Security Specifications. www.sevecom.org, 2009.

[59] G. Steel. Towards a formal security analysis of the Sevecom API. In *ESCAR*. 2009.

[60] G. Steel. Abstractions for verifying key management apis. In *SecReT*. 2010.

[61] C. Weidenbach. Towards an automatic analysis of security protocols. In H. Ganzinger, editor, *Proceedings of the 16th International Conference on Automated Deduction: CADE'99*, LNCS 1632, pages 378–382. Springer-Verlag, Berlin, 1999.

[62] C. Weidenbach, R. A. Schmidt, T. Hillenbrand, R. Rusev, and D. Topic. System description: Spass version 3.0. In *CADE*, pages 514–520, 2007.