Analysis of communicating authorization policies

ETHZ CS Technical Report #770. August 2012. To appear in the 8th Workshop on Security and Trust Management (STM 2012)

Analysis of Communicating Authorization Policies

Simone Frau and Mohammad Torabi Dashti

ETH Zürich

Abstract. We present a formal language for specifying distributed authorization policies that communicate through insecure asynchronous media. The language allows us to write declarative authorization policies; the interface between policy decisions and communication events can be specified using guards and policy updates. The attacker, who controls the communication media, is modeled as a message deduction engine. We give trace semantics to communicating authorization policies, and formulate a generic reachability problem. We show that the reachability problem is decidable for a large class of practically-relevant policies specified in our formal language.

1 Introduction

Ideally, by enforcing distributed authorization policies, the behavior of a distributed system should be constrained so that the distributed system achieves its functional goals without ever entering an insecure state. In a hospital, for example, a typical functional goal is to enable the medical personnel to access the health records of their patients; an instance of the unreachability of insecure states is: no one else can (ever) access these records. Decentralized trust management systems, or distributed authorization logics, play a pivotal role in securing distributed systems [1, 9, 13, 17, 24]. They allow us to formally reason about distributed authorization policies, even prior to their deployment.

We are, in this paper, concerned with policy decision points (PDPs) that communicate with each other by exchanging messages over insecure media. The policies of such PDPs change due to receive events (e.g. upon receiving a public key certificate), and they in turn constrain the communication events (e.g. access tokens are sent only to the principals whose credentials have not been revoked). As PDPs communicate over insecure media, attacks may occur when, for instance, expired certificates are replayed, certificate revocation lists are delayed, messages are tampered with, etc. We let the attacker be in direct control of the communication media. This view is motivated by the workings of security-sensitive distributed services, such as federated identity management systems (OAUTH [27], etc.). We define a formal language for specifying communicating authorization policies, and give an algorithm for deciding reachability for a large class of such policies. This builds upon the previous work on analyzing security-sensitive distributed services [8, 21], and the formalisms of [20, 26].

In our formalism, we model *communicating authorization policies* as a finite number of *processes*. Intuitively, a process represents a PDP. Each process consists of a finite number of *threads* that share a *policy*. Threads are finite sequences of communication events; they run in parallel and exchange messages with the threads of other processes over insecure media. The policy of a process is a (declarative) program which models the shared authorization policy that the threads of the process evaluate.

Threads communicate by sending and receiving messages, as is common in asynchronous message passing environments. Each send event is constrained by a *guard*, and each receive event results in an *update* of a process' policy. Intuitively, guards and updates belong to the *policy level*, as opposed to send and receive events which constitute the *communication level*. See Figure 1. From an operational point of view, guards are statements that, if derivable from the policy of a process, allow the process to perform a corresponding send action (cf. Dijkstra's guarded command language). Updates are also statements at the policy level. When a process receives a message in one of its threads, it updates its policy correspondingly. Intuitively, updates associate meanings to the messages a process receives in terms of statements at the policy level. For example, a signed X.509 certificate sent by a certificate authority *means* that the authority endorses the public key and its owner, mentioned in the certificate.

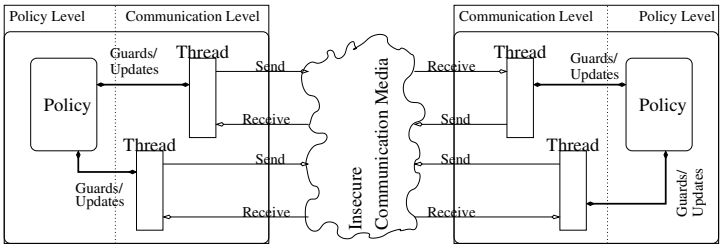

Fig. 1. Two processes (i.e. policy decision points), each with two threads.

We assume an all-powerful attacker who is in direct control of the insecure communication media; see Figure 1. In fact, the messages the PDPs exchange are passed through the attacker. This is a common (worst-case scenario) assumption in the literature. The message inference capabilities of the attacker may reflect, e.g., the Dolev-Yao threat model [18]. The attacker can indirectly manipulate the policies of the participating PDPs by, e.g., sending tampered messages which affect the update statements.

Contributions. Our main contributions are: **(1)** We present a formalism for specifying communicating authorization policies, and their hostile environment. **(2)** We give an algorithm for deciding reachability of communicating authorization policies. Given a formal model of the policy decision points and the attacker,

we can decide (under certain conditions, specified precisely in § 4) whether, or not, an (insecure) state is reachable. For instance, questions of the form *can the attacker learn the content of a certain file?*, or *can Ann ever be authorized to access a certain document stored at the file server?* are expressible as reachability problems. Note that communication level events, policy level computations, and the interface between the two are all taken into account when deciding reachability. This sets apart our decision algorithm from those for deciding secrecy in security protocols, and those for deciding reachability in dynamic authorization logics (see our related work, § 5).

The set of communicating authorization policies for which we can decide reachability is of practical relevance. We demonstrate this through a number of intuitive examples in the paper, and also by formalizing an electronic health record system, originally described in [3]. Our formalization can be found in the appendices.An implementation of our decision algorithm is publicly available at `http://www.infsec.ethz.ch/people/fraus`.

Structure of the paper. In § 2, we introduce the main features of our formal language with a few examples. The syntax and semantics of the language are formally defined in § 3. In § 4 we identify a set of communicating authorization policies for which the reachability problem is decidable. All proofs are relegated to the appendices. We conclude the paper in § 5 with reviewing the related work.

2 Examples of Communicating Authorization Policies

We introduce the main features of our formalism through three intuitive examples. The first example concerns an OAUTH 2.0 authorization endpoint that is constrained by an RBAC system with transitive attributes. The second example pertains to secure delegation of trust for distributing public key certificates. The third example discusses mechanisms for trust and permission revocation. The syntax and semantics of the language are formally defined, resp., in § 3.1 and § 3.2.

2.1 OAUTH, RBAC and Transitive Attributes

Suppose that Ann wants a remote print service, called RPS, to access (on her behalf) the file *brochure.pdf* stored on a file server. RPS uses OAUTH 2.0's Authorization Code Flow protocol [27] to point Ann to the file server where she can authenticate herself and grant RPS access to the file. After Ann authenticates herself to the file server, the file server needs to first check that Ann is authorized to access the file. If so, the file server sends a URI redirect message to Ann's Web browser, along with an authorization code. RPS can use the authorization code to access the file, after authenticating itself at a *token endpoint*. To keep the example simple, we do not model the entire scenario. Instead, we focus on the file server, which is an OAUTH 2.0 *authorization endpoint*.

Suppose that the file server is constrained by an RBAC policy with two roles, *user* and *admin*. The server stores two sorts of files: *public* and *confidential*. Any symbolic link to a public file is deemed public, and any link to a confidential file is deemed confidential. By transitivity, links to "links to confidential (resp. public) files" are confidential (resp. public). Users may access any public file or public link; admins may access any confidential file or link. Admins inherit all the rights attributed to users. Here is a formalization of the policy of the file server.

$$\mathcal{K}(\mathit{has\_role}(A, \mathit{user})) \leftarrow \mathcal{K}(\mathit{has\_role}(A, \mathit{admin}))$$
$$\mathcal{K}(\mathit{can\_access}(A, F)) \leftarrow \mathcal{K}(\mathit{has\_role}(A, \mathit{user})), \mathcal{K}(\mathit{has\_attrib}(F, \mathit{public}))$$
$$\mathcal{K}(\mathit{can\_access}(A, F)) \leftarrow \mathcal{K}(\mathit{has\_role}(A, \mathit{admin})), \mathcal{K}(\mathit{has\_attrib}(F, \mathit{confid}))$$
$$\mathcal{K}(\mathit{has\_attrib}(\mathit{link\_to}(F), X)) \leftarrow \mathcal{K}(\mathit{has\_attrib}(F, X))$$

The rules above are Horn clauses. We use capital letters to denote variables, and constants are written with lower-case letters. The predicate symbol $\mathcal{K}$, standing for *knows*, denotes the knowledge of the file server's policy engine, which "knows" pieces of information; e.g. the file server's policy engine knows that Ann has the role admin. The Horn clauses thereby model the server's information inference rules. For example, the first rule states that the file server knows that if A has the role admin, then A has the role user too, due to the role hierarchy. The second (resp. the third) rule above states that users (resp. admins) can read public (resp. confidential) files. The fourth rule states that links to public (resp. confidential) objects are themselves public (resp. confidential). We will often omit $\mathcal{K}$ to avoid unnecessary cluttering.

The extensional policy, or the extensional knowledge, of the file server reflects the "current state" of the policy. For example, Ann is an admin, and *brochure.pdf* is public:

$$\mathit{has\_role}(\mathit{ann}, \mathit{admin})$$
$$\mathit{has\_attrib}(\mathit{brochure.pdf}, \mathit{public})$$

Now, let us consider the communication between Ann and the file server. After Ann permits RPS to access *brochure.pdf*, the file server sends back an authorization code to Ann's *user agent*, e.g. her Web browser. The following guarded send event is part of the file server's thread:

$$\{\mathit{can\_access}(A, F)\}\{\}^{\neg} \blacktriangleright \mathsf{snd}(\mathit{auth\_code}(F), R\_URI, S\_INF)$$

The send event $\{s_1, ..., s_j\}\{n_1, ..., n_k\}^{\neg} \blacktriangleright \mathsf{snd}(m)$ can be executed only if $s_1, \cdots, s_j$ all follow from the policy, and at least one of $n_1, \cdots, n_k$ does not. That is, if the file server's policy implies $\mathit{can\_access}(\mathit{ann}, \mathit{brochure.pdf})$, then an authorization code for the file *brochure.pdf* is sent to Ann's browser. The authorization code is accompanied with a redirection URI R_URI and RPS's state information S_INFO. The user agent redirects Ann to RPS, which uses the authorization code to access the requested file after authenticating itself to a token endpoint. Note that, given the policy and extensional knowledge above, the file server can indeed derive $\mathit{can\_access}(\mathit{ann}, \mathit{brochure.pdf})$.

2.2 Trust Relations

Ann knows that the root certificate authority RCA is trusted on public keys. Ann asks RCA for Piet's public key. RCA is however temporarily overloaded, and redirects Ann's request to a local certificate authority CA. RCA trusts CA on public keys. CA sends back to Ann a public key of Piet. Here, we focus on two processes: Ann and CA.

The policy of the process CA contains all public keys belonging to Piet in our scenario. CA also stores a list of revoked public keys. For example,

$$\begin{array}{l} \mathit{is\_pk\_of}(\mathit{pk}_1, \mathit{piet}) \\ \mathit{is\_pk\_of}(\mathit{pk}_2, \mathit{piet}) \\ \mathit{is\_pk\_of}(\mathit{pk}_3, \mathit{piet}) \\ \mathit{is\_revoked}(\mathit{pk}_2) \end{array}$$

CA signs and sends out valid public keys upon request. The following thread, which is a sequence of two events, within the CA process models this behavior:

$$\begin{array}{l} \mathsf{rcv}(A, P) \blacktriangleright \{\}\{\}^{\neg} \\ \{\mathit{is\_pk\_of}(K, P)\}\{\mathit{is\_revoked}(K)\}^{\neg} \blacktriangleright \\ \qquad\qquad \mathsf{snd}(\mathit{sig}(\mathit{ca}, \mathit{pk\_cert}(P, K)), \mathit{sig}(\mathit{rca}, \mathit{is\_pk\_certified}(\mathit{ca}))) \end{array}$$

Here CA receives a pair of names A and P: A is asking CA for a public key of P. This message could have been redirected to CA by, e.g., RCA. Receiving this message binds the variables A and P in the rest of the thread. We remark that the events of the thread are executed sequentially. The message neither adds policy statements to CA's extensional knowledge, nor retracts any policy statements; hence the sequence $\{\}\{\}^{\neg}$ after rcv. We come back to this point shortly. Next, the CA sends out the message $\mathit{pk\_cert}(P, K)$, signed by CA, denoting that CA endorses K as a valid public key of P. The endorsement is accompanied by a message signed by RCA which certifies that RCA trusts CA on public keys $\big(\mathit{is\_pk\_certified}(\mathit{ca})\big)$. CA is assumed to have obtained the RCA-signed certificate prior to this exchange.

The CA sends out this message only if:

- CA's policy entails $\mathit{is\_pk\_of}(K, P)$, that is K is a public key of P, and
- the policy does *not* entail $\mathit{is\_revoked}(K)$, i.e. K has not been revoked.

The variable K, originating in the guard $\mathit{is\_pk\_of}(K, P)$, allows the CA to make a non-deterministic choice. In this example, the CA may choose to send either pk_1 or pk_3 as a valid public key for Piet. The key pk_2, however, cannot be sent, because it has been revoked according to CA's policy.

We turn to Ann's process. The thread that is pertinent to this scenario is given below:

$$\begin{array}{l} \{\}\{\}^{\neg} \blacktriangleright \mathsf{snd}(\mathit{ann}, \mathit{piet}) \\ \mathsf{rcv}(\mathit{sig}(X, \mathit{pk\_cert}(\mathit{piet}, K)), \mathit{sig}(\mathit{rca}, \mathit{is\_pk\_certified}(X))) \blacktriangleright \\ \qquad \{\mathit{said}(X, \mathit{is\_pk\_of}(K, \mathit{piet})), \mathit{said}(\mathit{rca}, \mathit{tdon}(X, \mathit{is\_pk\_of}(K, \mathit{piet})))\}\{\}^{\neg} \\ \{\mathit{is\_pk\_of}(K', \mathit{piet})\}\{\}^{\neg} \blacktriangleright \mathsf{snd}(\mathit{penc}(K', \mathit{payload})) \end{array}$$

Ann sends out a message (destined to RCA) asking for Piet's public key. The thread is tailored to handle delegation: Ann expects to receive a properly formatted message, signed by an arbitrary process X, given that X has RCA's trust on public keys. Receiving such a message affects the extensional knowledge of Ann. Ann adds two statements to her knowledge:

- $said(X, is\_pk\_of(K, piet))$ states that X endorses K as a public key of Piet. Note that is_pk_of in this process is local to Ann, i.e. it is independent of the internals of CA's policy, which happens to use the same function symbol for storing public keys and their owners.
- $said(rca, tdon(X, is\_pk\_of(K, piet)))$ states that RCA certifies that X is trusted on (denoted $tdon$) public keys; in particular, any public key associated to Piet.

In general, Ann may interpret a message either by adding policy statements to, or by retracting policy statements from, her extensional knowledge. That is, the receive event $\mathsf{rcv}(m) \blacktriangleright \{s_1, ..., s_j\}\{n_1, ..., n_k\}^{\neg}$ means that upon receiving message m, the statements $s_1, \cdots, s_j$ are all added to Ann's extensional knowledge, and $n_1, \cdots, n_k$ are all retracted from her extensional knowledge. Let us assume that CA sends the message $sig(ca, pk\_cert(piet, pk_3))$, $sig(rca, is\_pk\_certified(ca))$ to Ann. Then, Ann adds to her knowledge the following statements:

$$\begin{array}{ll} (s_1) & said(ca, is\_pk\_of(pk_3, piet)) \\ (s_2) & said(rca, tdon(ca, is\_pk\_of(pk_3, piet))) \end{array}$$

We remark that Ann does not retract any part of her extensional knowledge here. We come back to the notion of retraction in § 2.3.

Finally, Ann sends the message *payload* to Piet, encrypted with K', only if Ann knows that K' is Piet's public key (asymmetric encryption is denoted $penc(\cdot, \cdot)$).

With respect to Ann's policy, we have mentioned above that Ann does not directly trust CA on public keys. She trusts RCA, who delegates its rights w.r.t. endorsing public keys to CA. We model Ann's policy below, where rules are labeled for ease of reference.

$$\begin{array}{ll} (tr) & tdon(rca, is\_pk\_of(K, U)) \leftarrow \\ (td) & tdon(P, tdon(Q, X)) \leftarrow tdon(P, X) \\ (ta) & X \leftarrow said(P, X), tdon(P, X) \end{array}$$

The first rule (tr), with no preconditions, models Ann's trust relation: Ann knows that RCA is trusted on ($tdon$) any public key. The second rule (td) is the essence of transitive trust delegation: If P is trusted on X, then P can delegate this to Q. We remark that delegation of trust need not in general be transitive; see [24] for more on transitive and non-transitive delegation. The third rule (ta) describes trust application: If Ann knows that P said X, and she knows that P is trusted on X, then Ann can infer X; see, e.g., [24] for more details on trust application. Recall that the left-hand side of the last rule is in fact $\mathcal{K}(X)$; the predicate symbol $\mathcal{K}$ is suppressed.

Assuming that the statements (s_1) and (s_2) have been added to Ann's extensional knowledge, we show that Ann's policy would entail: pk_3 is Piet's public key. Thus Ann would send the message *payload* to Piet encrypted using pk_3. The following tree, annotated with the names of rules and statements used, shows Ann's deduction steps.

$$\dfrac{(s_1) \qquad \dfrac{(s_2) \quad \dfrac{\dfrac{}{tdon(rca, is\_pk\_of(pk_3, piet))}\,(tr)}{tdon(rca, tdon(ca, is\_pk\_of(pk_3, piet)))}\,(td)}{tdon(ca, is\_pk\_of(pk_3, piet))}\,(ta)}{is\_pk\_of(pk_3, piet)}\,(ta)$$

A typical security goal for this scenario is: *would Ann ever send a message encrypted with pk_2 (which has been revoked) to Piet?*. Another example: *for any key K, if Ann infers that K is Piet's public key, then Ann trusts RCA on 'K is Piet's public key'*. These goals can be expressed as reachability decision problems; cf. § 3.3. A side note: the scenario described above is susceptible to replay attacks, because there is no freshness guarantees for the certified keys received by Ann.

2.3 Retraction

Retracting policy statements is a feature often needed for modeling revocation of roles and permissions. Consider a hospital where a sensitive ward can be accessed only by the personnel who work in the ward, and have been vaccinated against a particular virus. The PDP that controls access to the ward would then contain a thread of the form:

$$\mathsf{rcv}(U, sig(ppc, is\_vaccinated(U))) \blacktriangleright \{vaccinated(U)\}\{\}^{\neg}$$
$$\{in\_ward(U), vaccinated(U)\}\{\}^{\neg} \blacktriangleright \mathsf{snd}(access\_token(U))$$

That is, an access token is sent to U only if U works in the ward (denoted $in\_ward(U)$), and U has been vaccinated. The statement that U has been vaccinated is added to the extensional knowledge of the PDP only if the hospital's personnel protection center PPC has put its signature on the message $is\_vaccinated(U)$.

Now, suppose the vaccination must be repeated every six months. The PDP would then need to inquire PPC about the status of U's vaccination: whether it is recent or not. Here is a (partial) specification of the thread in the PDP that contacts PPC:

$$\{\}\{\}^{\neg} \blacktriangleright \mathsf{snd}(v\_status(U))$$
$$\mathsf{rcv}(sig(ppc, has\_expired(X))) \blacktriangleright \{\}\{vaccinated(X)\}^{\neg}$$

That is, the PDP asks PPC about the vaccination status (v_status) of U. The PPC would send back to the PDP a signed message declaring that X's vaccination has been expired. PPC would let $X = U$ if U's vaccination has been

expired; otherwise, PPC lets X be a dummy name. Then, PDP retracts the fact $vaccinated(X)$ from its policy.

Two remarks are due. First, note that the messages exchanged between PDP and PPC do not have freshness guarantees. Therefore, the attacker can replay these messages, possibly misinforming the PDP about the freshness of the vaccination of the personnel. The second, and perhaps more important, point is that the thread of the PDP that issues access tokens does not synchronize with the PDP thread that contacts the PPC. Therefore, race conditions can arise (or, can be caused by a malicious scheduler) where the access token is issued for a personnel whose vaccination is no longer effective. The first problem can be solved using the standard challenge-response message exchange patterns. The second problem can be solved, e.g., using a lock inside the PDP policy engine. This point is further discussed below.

Synchronization. In order to enforce synchronization inside the PDP, we can use a shared lock between the two threads. This can be modeled as:

Thread1 :
$\mathsf{rcv}(U, sig(ppc, is\_vaccinated(U))) \blacktriangleright \{vaccinated(U), lock(U)\}\{\}^{\neg}$
$\{in\_ward(U), vaccinated(U)\}\{lock(U)\}^{\neg} \blacktriangleright \mathsf{snd}(access\_token(U))$
Thread2 :
$\{lock(U)\}\{\}^{\neg} \blacktriangleright \mathsf{snd}(v\_status(U))$
$\mathsf{rcv}(sig(ppc, has\_expired(X))) \blacktriangleright \{\}\{vaccinated(X), lock(U)\}^{\neg}$

Upon receiving a request from U, PDP's thread 1 locks the user U internally. The lock can be released only by thread 2 of the PDP who contacts the PPC to check the status of U's vaccination. The execution of thread 1 can then resume, only if thread 2 has not retracted the statement "U is vaccinated". [1]

Propagating Revocations. As mentioned above, retracting policy statements if used without the necessary synchronization mechanisms can be futile. This point is also relevant to how revocations are propagated in distributed settings. For instance, consider the file server of § 2.1. Remember that Ann has the role *admin* in the file server. Ann may therefore access any public file. Suppose that Ann's admin role is revoked after the file server has sent out the authorization code to RPS. After the revocation, Ann does not have the role *user* in the file server, because her user role was due to her admin role, which is revoked. The semantics of retraction in our formalism is *cascading* [12]: if a is entailed due to b, and only due to b, then retracting b would mean that a will not be entailed by the policy; see the semantics, § 3.2.

In short, the revocation is cascaded locally on the file server: all that followed from Ann being an admin is retracted after the revocation. However, RPS may

[1] This work does not directly address the problem of attribute staleness in attribute-based authorization logics in distributed settings, cf. [28]. This example shows, though, how our framework allows one to prevent staleness by means of synchronization mechanisms, or to detect consequent authorization violations otherwise.

use the authorization code issued by the file server to communicate with a token endpoint, hence accessing the file *brochure.pdf* on behalf of Ann. Meanwhile, Ann herself can no longer access the file. That is, in distributed settings, such as the scenario of § 2.1, revocations do not propagate automatically. To ensure that revocations take effect globally, synchronization mechanisms (e.g. through message passing, using shared objects) are in general necessary.

Careless uses, and unintentional effects, of retraction can be detected by checking reachability. For instance, in this example one could check that after the revocation Ann cannot access *brochure.pdf*, while a state in which RPS can access the file is reachable. Deciding reachability is thus crucial for understanding communicating policies.

(Un)Ambiguous Policies. A form of careless use of retraction is when the policy becomes ambiguous about rights. This point is best explained with an example: Suppose that the policy of a library service states that any student can access the reading groups schedule file *schedule.pdf*, formalized as:

$$\mathit{can\_access}(U, \mathit{schedule.pdf}) \leftarrow \mathit{is\_student}(U).$$

Now, imagine upon receiving a complaint about Ann, who is a student, the library service retracts her permission to access *schedule.pdf*:

$$\mathsf{rcv}(\mathit{complaint\_about}(U)) \blacktriangleright \{\}\{\mathit{can\_access}(U, \mathit{schedule.pdf})\}^{\neg}$$

The retraction is meant to prevent Ann from accessing the file, while the fact that she is a student implies that she may access the file. This is an ambiguity in the policy engine. In the semantics of our formalism (§ 3.2), the Horn rule trumps the retraction, because policies are seen as invariants of the process. That is, Ann would be able to access the file, despite "retracting" her access. For enforcing the retraction, either the fact $\mathit{is\_student}(U)$ should be retracted (which would be too harsh a reaction to an unsigned complaint), or the guard for accessing the file should be refined to explicitly exclude those about whom a complaint has been received.

We remark that retractions in our formalism respect the semantics of Griffiths and Wade [23]: if a policy statement is added to the extensional knowledge of a process, and subsequently the same statement is retracted, the policy remains unchanged. Due to Horn rules, however, retracting policy statements that do not belong to the extensional knowledge is inconsequential, as the example above shows. Adding and retracting policy statements, through updates, indeed affect only the extensional knowledge of the processes; see § 3.2.

3 Formalizing Communicating Authorization Policies

We formally define the syntax (§ 3.1) and semantics (§ 3.2) of our language. We also define a generic reachability problem (§ 3.3) for communicating authorization policies.

3.1 Syntax

A (first-order) *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, and $\mathcal{P}$ is a finite set of predicate symbols. 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 in $\mathcal{V}$, is denoted $\mathcal{T}_{\Sigma(\mathcal{V})}$. A *message* is a ground term, i.e. an element of $\mathcal{T}_{\Sigma(\emptyset)}$. The set of *atoms* is defined as $\{p(t_1, \cdots, t_n) \mid p \in \mathcal{P}, t_i \in \mathcal{T}_{\Sigma(\mathcal{V})},$ arity of p is $n\}$. A *fact* is an atom with no variables. In the following we fix a signature $(\Sigma, \mathcal{V}, \mathcal{P})$.

An *event* is either a *send event* or a *receive event*. A send event is of the form $g \blacktriangleright \mathsf{snd}(m)$, where g is a *guard* and m is a term. A receive event is of the form $\mathsf{rcv}(m) \blacktriangleright u$, where m is a term and u is an *update*. A guard g is of the form $g_\exists\ g_{\not\exists}$, where $g_\exists$ and $g_{\not\exists}$ are disjoint finite sets of atoms. An update u is of the form $u_+\ u_-$, where u_+ and u_- are disjoint finite sets of atoms. We refer to $g_\exists$ (resp. u_+) as a positive guard (resp. update), and to $g_{\not\exists}$ (resp. u_-) as a negative guard (resp. update). In the examples of § 2 we have superscripted negative guards and negative updates with $\neg$ to improve the readability of the specifications.

A thread t is a finite sequence of events $t = e_1, \cdots, e_n$, $n \geq 0$. For a variable v appearing in thread t, we say the event e_i, with $1 \leq i \leq n$, is the origin of v in t iff i is the smallest index of the events in which v appears. In the remainder of the paper we only consider threads t that satisfy the *origination property*: for any variable v appearing in t the following holds.

- if the origin of v is the event $\mathsf{rcv}(m) \blacktriangleright u_+\ u_-$, then either v appears in m, or v appears in u_- and v does not appear elsewhere in the thread;
- if the origin of v is the event $g_\exists\ g_{\not\exists} \blacktriangleright \mathsf{snd}(m)$, then either v appears in $g_\exists$, or v appears in $g_{\not\exists}$ and v does not appear elsewhere in the thread.

A *process* π is a tuple $(\eta_\pi, \Omega_\pi, \mathbf{I}_\pi)$, where η_π is a finite set of threads, Ω_π is a finite set of facts, called the *extensional knowledge* or *extensional policy* of π, and $\mathbf{I}_\pi$ is a finite set of *Horn clauses*, called the *policy* of π. 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.

An *attacker model* $\mathbb{A}$ is a pair $(\Omega_\mathbb{A}, \vdash^\mathbb{A})$, where $\Omega_\mathbb{A}$ is a finite set of messages, called the extensional knowledge of the attacker, and $\vdash^\mathbb{A}$, referred to as the policy of the attacker, is a subset of $2^{\mathcal{T}_{\Sigma(\emptyset)}} \times \mathcal{T}_{\Sigma(\emptyset)}$; intuitively, $M \vdash^\mathbb{A} m$ means that the attacker can derive message m, given the finite set M of messages. The attacker is thus identified in our model with her extensional knowledge (which stores the set of the messages observed by the attacker) and her policy (which encodes the deduction capabilities of the attacker). Moreover, the attacker is able to send and receive messages; these capabilities are reflected in the execution model, defined in § 3.2.

We are now ready to define communicating authorization policies, hereafter referred to as CAPs. A CAP is a tuple $((\Sigma, \mathcal{V}, \mathcal{P}), \pi_1, \cdots, \pi_\ell, \mathbb{A})$, where $(\Sigma, \mathcal{V}, \mathcal{P})$ is a signature, $\pi_1, \cdots, \pi_\ell$ are (honest) processes and $\mathbb{A}$ is an attacker model, all defined using the signature $(\Sigma, \mathcal{V}, \mathcal{P})$. To avoid trivial name clashes, variables appearing in different threads are assumed to be distinct.

3.2 Semantics

For a finite set H of Horn clauses, a fact a and a finite set T of facts, we write $T \vdash^H a$ iff a belongs to the closure of T under H. We write $T \vdash^H T'$, for a finite set of facts T', iff $T \vdash^H a$ for all $a \in T'$. Let A be a set of atoms. Define $A{\downarrow}$ as the set of all facts, i.e. ground atoms, that can be obtained by applying a substitution to the elements of A.

Given a process $\pi = (\eta_\pi, \Omega_\pi, \mathbf{I}_\pi)$, we use $\vdash^\pi$ as a shorthand for $\vdash^{\mathbf{I}_\pi}$, and we refer to the set $\{a \mid \Omega_\pi \vdash^\pi a\}$ as the *knowledge* of π. That is, the knowledge of π is the set of facts that can be inferred from the extensional knowledge of π using its policy. Similarly, the knowledge of the attacker $(\Omega_\mathbb{A}, \vdash^\mathbb{A})$ is the set $\{m \mid \Omega_\mathbb{A} \vdash^\mathbb{A} m\}$.

A *configuration* of $\mathsf{cap} = ((\Sigma, \mathcal{V}, \mathcal{P}), \pi_1^0, \cdots, \pi_\ell^0, \mathbb{A}^0)$ is an $\ell + 1$ tuple where the first ℓ elements are processes, and the last element is an attacker model. The *initial* configuration of cap is the tuple $(\pi_1^0, \cdots, \pi_\ell^0, \mathbb{A}^0)$. We define trace semantics for CAPs. Let Z be the set of all configurations of cap, i.e. the set of all tuples of the form $z = (\pi_1, \cdots, \pi_\ell, \mathbb{A})$. We define the relation $\rightarrow \subseteq Z \times E \times Z$ as: $(z, e, z') \in \rightarrow$ iff $z = (\pi_1, \cdots, \pi_i, \cdots, \pi_\ell, \mathbb{A})$, $z' = (\pi_1, \cdots, \pi_i', \cdots, \pi_\ell, \mathbb{A}')$, $\mathbb{A} = (\Omega_\mathbb{A}, \vdash^\mathbb{A})$, $\pi_i = (\eta, \Omega, \mathbf{I})$ and $p = e \cdot p' \in \eta$ for some thread p' and $1 \leq i \leq \ell$, and one of the following conditions hold:

- $e = g_\exists\ g_\nexists \blacktriangleright \mathsf{snd}(t)$, $\mathbb{A}' = (\Omega_\mathbb{A} \cup \{t\sigma\}, \vdash^\mathbb{A})$ and $\pi_i' = (\eta \setminus \{p\} \cup \{p'\sigma\}, \Omega, \mathbf{I})$, for some substitution σ that satisfies the following conditions:
  - $g_\exists\sigma$ is a set of facts and $\Omega \vdash^{\pi_i} g_\exists\sigma$.
  - If $g_\nexists$ is a non-empty set, then there exists no substitution σ' s.t. $\Omega \vdash^{\pi_i} g_\nexists\sigma\sigma'$.
- $e = \mathsf{rcv}(t) \blacktriangleright u_+\ u_-$, $\mathbb{A}' = \mathbb{A}$ and $\pi_i' = (\eta \setminus \{p\} \cup \{p'\sigma\}, \Omega \setminus u_-\sigma{\downarrow} \cup u_+\sigma, \mathbf{I})$, for some substitution σ such that $t\sigma$ is a message and $\Omega_\mathbb{A} \vdash^\mathbb{A} t\sigma$.

We write $z \xrightarrow{e} z'$ for $(z, e, z') \in \rightarrow$. A *trace* of cap is an alternating sequence of configurations and events of the form $z_0 e_1 z_1 \cdots z_{n-1} e_n z_n$, with $n \geq 0$, such that z_0 is the initial configuration of cap and $z_{i-1} \xrightarrow{e_i} z_i$ for all $i \in [1, n]$. The *semantics* of the cap is defined as the set of all its traces. We say that configuration z is *reachable* in cap iff there exists a trace $z_0 e_1 z_1 \cdots e_n z_n$ in the semantics of cap with $n \geq 0$ and $z = z_n$.

Note that the extensional knowledge of the processes and the attacker in any reachable configuration of cap are indeed sets of facts (i.e. they do not contain variables). This is due to the origination property and the conditions the semantics enforces on substitutions. Furthermore, the transition relation defined above does not change the policies of the processes or the attacker models (only threads and extensional knowledge change).

An interesting feature of our formalism, not explained in the examples of § 2, is the variables that originate in negative guards or updates. This feature is described below.

Example 1 (Variables originating in negative guards or updates, and SoD). Consider a hospital data center HDC that receives a message from the hospital's human resources department HR stating that a doctor has retired. The following

thread of HDC models this event:

$$\mathsf{rcv}(sig(hr, retired(D))) \blacktriangleright \{\}\{patient\_of(P, D)\}^{\neg}$$

That is, after receiving the message, HDC retracts the statements $patient\_of(P, D)$ for *all* patients P. In other words, after D has retired, no patient can be considered as his/her patient. That is, the variables that originate in negative updates are interpreted universally (see the semantics). It is therefore natural that such variables cannot be referred to in the rest of the thread; see the origination property. We remark that variables cannot originate in positive updates.

Now we turn to the variables that originate in negative guards. Suppose that HDC receives a report R from a medical personnel P. To review the report, the report may be forwarded to any doctor D who does not work in the same ward as P. The following thread models this scenario.

$$\begin{aligned}&\mathsf{rcv}(penc(pk(hdc), sig(P, report(R)))) \blacktriangleright \{\}\{\}^{\neg}\\ &\{is\_doctor(D)\}\{in\_ward(P, W), in\_ward(D, W)\}^{\neg} \blacktriangleright\\ &\qquad\qquad \mathsf{snd}(penc(pk(D), please\_review(R)))\end{aligned}$$

Here, $in\_ward(X, W)$ means that X works in ward W, $pk(X)$ is a public key of process X. The variable W originates in a negative guard. Therefore, according to the semantics given above, this guard is satisfied for any D such that: (as in § 2, we omit the predicate symbol $\mathcal{K}$, i.e. *knows*, around these terms to simplify the presentation)

$$is\_doctor(D) \wedge \nexists W.\ in\_ward(P, W) \wedge in\_ward(D, W)$$

The guard is satisfied for D only when there is *no* instantiation for W where both D and P work in ward W. Variables originating in negative guards can therefore not be referred to elsewhere in the thread; see the origination property.

Similarly, separation of duty (SoD) can be expressed in our formalism. Two tasks T_1 and T_2 are constrained under an SoD relation iff no single principal is allowed to perform them both. In general, SoD relations must be anti-reflexive and symmetric. Consider the following guarded send event:

$$\{can\_do(A, T_1)\}\{has\_done(A, T_2), in\_sod(T_1, T_2)\}^{\neg} \blacktriangleright \mathsf{snd}(auth\_token(A, T_1))$$

An authorization token to perform task T_1 is issued for A only if

- A is allowed to perform T_1, denoted $can\_do(A, T_1)$.
- There exists no task T_2 such that T_2 has been performed by A, denoted $has\_done(A, T_2)$, and there is a SoD constraint on tasks T_1 and T_2, denoted $is\_sod(T_1, T_2)$. That is,

$$can\_do(A, T_1) \wedge \nexists T_2.\ has\_done(A, T_2) \wedge in\_sod(T_1, T_2)$$

In case the authorization code to perform T_1 is sent to A, the fact $has\_done(A, T_1)$ should be added (via update statements) to the extensional knowledge of the process that enforces SoD, after A has performed the task. This is necessary to keep a complete record of the tasks that have been performed.

It is easy to observe that changes in the SoD relation can be expressed using positive and negative updates in our formalism. (end of example)

3.3 The Reachability Decision Problem

Fix a CAP, defined as $\mathsf{cap} = ((\Sigma, \mathcal{V}, \mathcal{P}), \pi_1, \cdots, \pi_\ell, \mathbb{A})$. A *goal* G for cap is an $\ell + 1$ tuple $G = (f_1, \cdots, f_\ell, m)$, where $f_i \in \mathcal{A}_{\Sigma(\emptyset)}$ and $m \in \mathcal{T}_{\Sigma_{msg}(\emptyset)}$. Intuitively, goals are conditions on the knowledge of the processes and the attacker: the knowledge of π_i must entail f_i, and m should belong to the knowledge of the attacker. The *reachability* problem $\textsc{Reach}\langle \mathsf{cap}, G \rangle$ asks whether, or not, there exists a reachable configuration $((\eta_1, \Omega_1), \cdots, (\eta_\ell, \Omega_\ell), \Omega_{\mathbb{A}})$ in cap such that

$$\forall i \in [1, \ell].\ \Omega_i \vdash^i f_i \bigwedge \Omega_{\mathbb{A}} \vdash^{\mathbb{A}} m$$

These cases are, resp., denoted by $\textsc{Reach}\langle \mathsf{cap}, G \rangle = \mathsf{T}$ and $\textsc{Reach}\langle \mathsf{cap}, G \rangle = \mathsf{F}$. A few remarks are due:

1. As a convention, we may write $\textsc{Reach}\langle \mathsf{cap}, \pi : f \rangle$ or $\textsc{Reach}\langle \mathsf{cap}, \mathbb{A} : m \rangle$ when we are interested only in the knowledge of process π, or the knowledge of the attacker $\mathbb{A}$. This can be obviously reduced to the reachability problem defined above, by adding dummy facts/messages to the knowledge of the processes/attacker whose knowledge is of no interest to us. For example, take the decision problem that asks whether, or not, a file server, represented by process π, may ever authorize Ann to download a file; we write a for the ground atom that in π's policy represents Ann's right to download the file. This decision problem can be encoded as $\textsc{Reach}\langle \mathsf{cap}, \pi : a \rangle$ in our formalism.
2. The decision problem REACH subsumes the *secrecy problem* for security protocols (with a bounded number of sessions). The secrecy problem asks whether the attacker can learn a supposedly secret message m through interacting with honest processes, e.g. see [30]. This problem can be represented as $\textsc{Reach}\langle \mathsf{cap}, \mathbb{A} : m \rangle$.
3. Reachability for guards can in general be reduced to REACH. Suppose we want to know whether, or not, cap reaches a state where $\Psi = \exists X.\ f(X) \wedge \nexists Y.\ g(X, Y)$ is satisfied in the policy of process π. This corresponds to checking whether the guard $\{f(X)\}\{g(X, Y)\}^\neg$ will ever be satisfied in process π. We can simply add to π a "helper" thread which consists of a single guarded send event:

   $$\{f(X)\}\{g(X, Y)\}^\neg \blacktriangleright \mathsf{snd}(\mathit{fresh\_term})$$

   where *fresh_term* is a message not appearing anywhere else in cap. Now, Ψ is ever satisfied in the policy of π iff $\textsc{Reach}(\mathsf{cap}, \mathbb{A} : \mathit{fresh\_term}) = \mathsf{T}$. [2]

Due to the semi-decidability of general Horn clauses, REACH is in general undecidable. In § 4 we identify a set of CAPs for which REACH is decidable.

[2] This reduction hinges on the assumption that if $m \in M$, then $M \vdash^{\mathbb{A}} m$. This assumption invariably holds for all the attacker models existing in the literature.

4 A Class of Decidable Communicating Authorization Policies

In this section, we identify a set of CAPs specified in our formal language for which REACH is decidable. We refer to this set as **DC**. Intuitively, a CAP belongs to **DC** iff the following conditions hold:

1. The policies of honest processes are written in a fragment of Horn theories, hereafter called **AL**, formally defined below. **AL** stands for *authorization logic*.
2. The attacker has the capabilities of the standard Dolev-Yao (DY) attacker [18]. The Dolev-Yao attacker intercepts and remembers all transmitted messages, it can encrypt, decrypt and sign messages if it knows the corresponding keys, it can compose and send new messages using the messages it has observed, and can remove or delay messages in favor of others being communicated.

First, we introduce the notion of *infons*, adopted from [24, 25]. Infons are pieces of information, e.g. $\mathit{can\_read}(\mathit{piet}, \mathit{file12})$ stipulating that Piet can read a certain *file12*. An infon does not admit a truth value, i.e. it is never false or true. Instead, infons are the interface between the communication level and policy level for honest processes. That is, if the policy of Ann entails the fact $\mathcal{K}(\mathit{can\_read}(\mathit{piet}, \mathit{file12}))$, then the process Ann "knows" that Piet may read *file12*, and she may thus grant him read access to *file12*. The simplest form of an infon is constructed by applying *wrappers* to message terms. For example, if the message term n is a nonce, then the wrapper *is_fresh* can be used to construct the piece of information $\mathit{is\_fresh}(n)$, denoting that the nonce n is fresh. More interesting infons are constructed by applying *infon constructor* functions to other infons (and messages). For example, $\mathit{said}(\pi, \mathit{is\_fresh}(n))$ is an infon that states that process π said that n is a fresh nonce. Then, $\mathit{said}(\pi', \mathit{said}(\pi, \mathit{is\_fresh}(n)))$ is the infon that states that process π' said that: process π said that n is fresh.

In the following, for any signature $(\Sigma, \mathcal{V}, \mathcal{P})$ we assume $\Sigma = \Sigma_{msg} \sqcup \Sigma_{infon}$ and $\mathcal{V} = \mathcal{V}_{msg} \sqcup \mathcal{V}_{infon}$, where Σ_{msg} is the set of message constructor functions, and Σ_{infon} is the set of infon constructors. The notation $\sqcup$ stands for the union of disjoint sets. Two types of terms are generated by Σ: message terms, and infons (defined formally below). We assume that the elements of Σ_{infon} are associated with types: each function symbol $f \in \Sigma_{infon}$ of arity n specifies the types of its n arguments.

- The elements of $\mathcal{T}_{\Sigma_{msg}(\mathcal{V}_{msg})}$ are called message terms, and have the type msg.
- The set *Infons* is the smallest set that contains $\mathcal{V}_{infon}$ and s.t. $f(t_1, \cdots, t_n) \in \mathit{Infons}$ for any $f \in \Sigma_{infon}$ of arity n, and t_i being of the correct type. The elements of *Infons* have the type infon.

An infon constructor function whose arguments are all of type msg is called a wrapper function. Note that the types msg and infon are disjoint. Any element of $\mathcal{T}_{\Sigma(\mathcal{V})}$ that is not a message or infon is considered ill-typed, hence ignored in our study.

We introduce the **AL** theories in order to model the policies of honest processes. [3] In **AL**, the policy statements for honest processes (i.e. facts derived from the processes' policies) are predicates over infons; i.e. for honest processes, the knowledge ranges over pieces of information.

Definition 1. *Take any finite set T of Horn clauses, defined over the signature $(\Sigma, \mathcal{V}, \mathcal{P})$, where any clause in T is of the form $p(t) \leftarrow p_1(t_1), \cdots, p_\ell(t_\ell)$, with $\ell \geq 0$, $p, p_1, \cdots, p_\ell \in \mathcal{P}$, and $t, t_1, \cdots, t_\ell \in$ Infons.*

T is an **AL** *theory iff $T = T_{\leftarrow} \sqcup T_{\rightarrow}$ where*

(a) The set rewrite system $\mathfrak{R}_{T_{\leftarrow}} = \{a \Rightarrow a_1 \cdots a_n \mid a \leftarrow a_1, \cdots, a_n \in T_{\leftarrow}\}$ induced by $T_{\leftarrow}$ is terminating, i.e. it does not admit reduction sequences of infinite length; cf. [4] for more on set rewrite systems.

(b) For each clause $p(t) \leftarrow p_1(t_1), \cdots, p_\ell(t_\ell)$ in $T_{\rightarrow}$, at least one of the t_j, with $j \in [1, \ell]$, is an anchor *for the clause; that is:*

- *$var(t_j)$ contains all the variables appearing in the clause, and*
- *all the variables in t_j are infon variables (i.e. elements of $\mathcal{V}_{infon}$), and*
- *$p_j(t_j)$ unifies with no atom appearing in T, except for itself and the atoms in $\{h \mid h \leftarrow t_1, \cdots, t_n \in T_{\rightarrow}\}$.*

Furthermore, the term rewriting system $\mathfrak{R}_{T_{\rightarrow}} = \{a \Rightarrow t \mid t \leftarrow t_1, \cdots, t_n, a \in T_{\rightarrow}, a$ is the anchor of $t \leftarrow t_1, \cdots, t_n, a\}$ is terminating.

It is easy to observe that all the policies defined in § 2 indeed belong to **AL**: for the examples of sections 2.1 and 2.3, all the rules belong to $T_{\leftarrow}$, while for the the example of § 2.2, T is partitioned into $T_{\leftarrow} = \{tr, td\}$ and $T_{\rightarrow} = \{ta\}$. For more detail, see the appendices.

Below, we define a set **DC** of CAPs for which we prove REACH is decidable.

Definition 2. *The set* **DC** *consists of all the* CAPs *$((\Sigma, \mathcal{V}, \mathcal{P}), \pi_1, \cdots, \pi_\ell, \mathbb{A})$ that satisfy the following conditions:*

- *$(\Sigma, \mathcal{V}, \mathcal{P})$ is a signature, and:*
  - *$\Sigma = \Sigma_{msg} \sqcup \Sigma_{infon}$, where $\sqcup$ denotes union of disjoint sets.*
  - *Σ_{msg} contains the functions $penc(\cdot,\cdot)$, $\{\!|\cdot|\!\}_\cdot$, $sig(\cdot,\cdot)$, $\mathsf{pk}(\cdot)$, $\mathsf{sk}(\cdot)$, $h(\cdot)$, $(\cdot,\cdot)$, representing the usual cryptographic primitives. Namely, they represent respectively asymmetric and symmetric encryption, digital signature, public key and private key constructors, hash and pairing functions.*
- *For any process $\pi = (\eta^0_\pi, \Omega^0_\pi, \mathbf{I}_\pi)$:*
  - *$\mathbf{I}_\pi$ is an* **AL** *theory.*
  - *Atoms in Ω^0_π, in guards and in updates are of the form $p(i)$, where $p \in \mathcal{P}$ and i is an infon. That is, the knowledge of the process ranges over infons.*
  - *Variables in the guards and updates range over messages, i.e. they are always under the application of a wrapper function symbol.*

[3] In contrast to Datalog, a fragment of Horn theories which has been extensively used in access control, **AL** allows for nested function symbols and variables that only appear in rules' heads.

- *In all the events* $g \blacktriangleright \mathsf{snd}(m)$ *and* $\mathsf{rcv}(m) \blacktriangleright u$, $m \in \mathcal{T}_{\Sigma_{msg}(\mathcal{V})}$. *This condition ensures that participants send and receive message terms, as opposed to infons.*

– $\mathbb{A} = (\Omega^0_{\mathbb{A}}, \vdash^{\mathbb{A}})$, *where* $\Omega^0_{\mathbb{A}}$ *is a finite subset of* $\mathcal{T}_{\Sigma_{msg}(\emptyset)}$, *and* $\vdash^{\mathbb{A}}$ *reflects the capabilities of the standard Dolev-Yao attacker, defined, e.g., in [32].*

If we consider the Dolev-Yao attacker for the examples of § 2, they all fall into **DC**.

The proof of the following theorem can be found in the appendices.

Theorem 1. $\textsc{Reach}\langle \mathsf{cap}, G\rangle$ *is decidable for any* cap *in* **DC**, *and any goal* G. *Moreover, if* $\textsc{Reach}\langle \mathsf{cap}, G\rangle = \mathsf{T}$, *then the decision algorithm returns a witness trace.*

5 Related Work

The closest related works are (1) dynamic authorization logics, and (2) security protocols annotated with authorization constraints.

(1) Dynamic authorization logics. Existing distributed authorization logics, such as [1,9,13,17,24], cannot express the dynamic aspects of distributed policies. We, in contrast, model communicating authorization policies which change due to communication events. Our formalism allows therefore for modeling the dynamic aspects of distributed (i.e. communicating) authorization policies. This is the fundamental difference between our formalism and the dynamic authorization logics that confine their analysis to a centralized policy decision point, such as [6, 14, 19, 22, 29].

We decide reachability in communicating authorization policies by taking into account the "low-level" cryptographic protocols that implement the policies, and also the interface between the low-level protocols and the policies. This is in contrast to the dynamic authorization logics that model the causes and effects of policy changes at the same level of abstraction as the policy, such as [10, 11]. These logics hence abstract away the mechanisms through which policy decision points communicate and possibly influence each other. For instance, the notion of updates in our language is close to the notion of *effects* in [10,11]. We however do not associate effects to policy decisions; rather, effects (or updates) are associated with receiving messages. What we call a guarded send event in our formalism has no counterpart in [10, 11]. In our decision algorithm, we not only take into account the asynchronous communications among the policy decision points, but also we can explicitly model the capabilities of the hostile entity (e.g. the Dolev-Yao [18] attacker) that controls their communications.

In [31], the authors use a variant of deontic logic to reason about changes in authorization policies. They assume that the policies are explicitly given as a finite-state Kripke model. We, in contrast, model systems that generally induce infinite state spaces.

(2) Security protocols annotated with authorization constraints. Our formalism is related to the body of research on annotating security protocols with authorization constraints [2, 7, 8, 20, 26]. None of these works give an algorithm for deciding reachability. In contrast, we can decide reachability for a large class of authorization policies.

Here, we extend our previous results [21] in two directions: **(1)** We have negative statements in guards, and we allow the policy statements to be retracted from the extensional policies of the processes. The negative guards and the retraction of policy statements enable us to naturally model non-monotonic behaviors, such as revocation of rights. Non-monotonic policy evaluations, if used without caution, can however lead to subtle attacks: the attacker may be able to access a security-sensitive object by withholding a certain credential. We discuss, within an example, how our decision algorithm can help policy writers to detect these situations; see § 2.3. **(2)** We give a decision algorithm for a rich set of policies, which strictly subsumes the *type-1* policy theories of [21]. For instance, transitive policy rules (see § 2.1), which are not permitted in type-1 theories, can be analyzed using our decision algorithm.

Acknowledgements. We would like to thank Samuel Burri, Srdan Marinovic and Petar Tsankov for their helpful comments on earlier drafts. We are grateful to Jorge Cuéllar for his help with our case study on electronic health records. This work is partially supported by NESSoS, the EU FP7-ICT-2009.1.4 Project No. 256980.

References

1. Martín Abadi, Michael Burrows, Butler Lampson, and Gordon Plotkin. A calculus for access control in distributed systems. *ACM Trans. Program. Lang. Syst.*, 15(4):706–734, 1993.
2. A. Armando et al. The AVANTSSAR platform for the automated validation of trust and security of service-oriented architectures. In *TACAS 2012*, volume 7214 of *LNCS*, pages 267–282. Springer, 2012.
3. AVANTSSAR. Deliverable D5.1: Problem cases and their trust and security requirements. Available at `http://www.avantssar.eu`, 2008.
4. F. Baader and T. Nipkow. *Term rewriting and all that.* Cambridge University Press, 1998.
5. Bahareh Badban, Jaco van de Pol, Olga Tveretina, and Hans Zantema. Generalizing DPLL and satisfiability for equalities. *Inf. Comput.*, 205(8):1188–1211, 2007.
6. A. Bandara, E. Lupu, and A. Russo. Using event calculus to formalise policy specification and analysis. In *POLICY '03*, pages 26–. IEEE CS, 2003.
7. Chetan Bansal, Karthikeyan Bhargavan, and Sergio Maffeis. Discovering concrete attacks on website authorization by formal analysis, 2012. to appear in CSF 2012.
8. M. Barletta, S. Ranise, and L. Viganò. A declarative two-level framework to specify and verify workflow and authorization policies in service-oriented architectures. *Service Oriented Computing and Applications*, 5(2):105–137, 2011.
9. M. Becker, C. Fournet, and A. Gordon. Design and semantics of a decentralized authorization language. In *CSF '07*, pages 3–15. IEEE Computer Society, 2007.

10. M. Becker and S. Nanz. A logic for state-modifying authorization policies. *ACM Trans. Inf. Syst. Secur.*, 13(3), 2010.
11. Moritz Y. Becker. Specification and analysis of dynamic authorisation policies. In *CSF*, pages 203–217. IEEE Computer Society, 2009.
12. E. Bertino, P. Samarati, and S. Jajodia. An extended authorization model for relational databases. *IEEE Trans. Knowl. Data Eng.*, 9(1):85–101, 1997.
13. Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized trust management. In *IEEE Symposium on Security and Privacy*, pages 164–173. IEEE CS, 1996.
14. A. Chaudhuri, P. Naldurg, S. Rajamani, G. Ramalingam, and L. Velaga. EON: modeling and analyzing dynamic access control systems with logic programs. In *ACM CCS '08*, pages 381–390. ACM, 2008.
15. H. Comon-Lundh, V. Cortier, and E. Zalinescu. Deciding security properties for cryptographic protocols. application to key cycles. *ACM Trans. Comput. Log.*, 11(2), 2010.
16. R. Corin and S. Etalle. An improved constraint-based system for the verification of security protocols. In *SAS '02*, volume 2477 of *LNCS*, pages 326–341. Springer, 2002.
17. J. DeTreville. Binder, a logic-based security language. In *IEEE S&P*, page 105, 2002.
18. D. Dolev and A. Yao. On the security of public key protocols. *IEEE Trans. on Information Theory*, IT-29(2):198–208, 1983.
19. D. Dougherty, K. Fisler, and S. Krishnamurthi. Specifying and reasoning about dynamic access-control policies. In *IJCAR '06*, pages 632–646. Springer, 2006.
20. C. Fournet, A. Gordon, and S. Maffeis. A type discipline for authorization policies. *ACM Trans. Program. Lang. Syst.*, 29(5), 2007.
21. S. Frau and M. Torabi Dashti. Integrated specification and verification of security protocols and policies. In *CSF*, pages 18–32. IEEE CS, 2011.
22. Deepak Garg and Frank Pfenning. Stateful authorization logic: proof theory and a case study. In *Security and Trust Management (STM '10)*, pages 210–225. Springer, 2011.
23. P. Griffiths and B. Wade. An authorization mechanism for a relational database system. *ACM Trans. Database Syst.*, 1(3):242–255, 1976.
24. Y. Gurevich and I. Neeman. DKAL: Distributed-knowledge authorization language. In *CSF '08*, pages 149–162. IEEE Computer Society, 2008.
25. Y. Gurevich and I. Neeman. The logic of infons. *Bulletin of the EATCS*, 98:150–178, 2009.
26. J. Guttman, F. Thayer, J. Carlson, J. Herzog, J. Ramsdell, and B. Sniffen. Trust management in strand spaces. In *ESOP '04*, volume 2986 of *LNCS*, pages 325–339, 2004.
27. E. Hammer et al. The OAuth 2.0 authorization framework, 2012. IETF.
28. Ram Krishnan, Jianwei Niu, Ravi S. Sandhu, and William H. Winsborough. Stale-safe security properties for group-based secure information sharing. In Vitaly Shmatikov, editor, *FMSE*, pages 53–62. ACM, 2008.
29. Ninghui Li and Mahesh V. Tripunitara. Security analysis in role-based access control. *ACM Trans. Inf. Syst. Secur.*, 9(4):391–420, November 2006.
30. J. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *CCS '01*, pages 166–175. ACM Press, 2001.
31. Riccardo Pucella and Vicky Weissman. Reasoning about dynamic policies. In *FoSSaCS '04*, volume 2987 of *LNCS*, pages 453–467. Springer, 2004.
32. M. Rusinowitch and M. Turuani. Protocol insecurity with finite number of sessions is NP-complete. In *CSFW '01*, page 174. IEEE CS, 2001.

A Policies of Examples in § 2 Belong to AL

For the examples of sections 2.1 and 2.3, we note that all the rules belong to $T_{\leftarrow}$. For the the example of § 2.2, however partitioning T to $T_{\leftarrow}$ and $T_{\rightarrow}$ is important:

- Let $T_{\leftarrow}$ consist of all the rules $p(t) \leftarrow p_1(t_1), \cdots, p_\ell(t_\ell)$ where t contains all the variables appearing in $t_1, \cdots, t_\ell$. We observe that the induced reduction system for $T_{\leftarrow}$ is indeed terminating. In particular, for the trust delegation rule
$$tdon(P, tdon(Q, X)) \leftarrow tdon(P, X)$$
note that in each reduction step the number of *tdon* infon constructors is reduced by one; hence termination.
- Let $T_{\rightarrow}$ contain all the other rules in T. That is, $T_{\rightarrow}$ consists of the trust application (ta) rule. To find an anchor for each rule of $T_{\rightarrow}$ we notice that anchors can only be produced by the clause in which they appear. This is because anchors do not unify with the atoms appearing in other rules of T. Therefore, anchors must be "initially" introduced when a (signed, and properly formatted) message is interpreted as a policy statement. Naturally, anchors can contain nested terms; hence the quotation mark around "initially".
  Following this intuition, we observe that the atom that contains *said* is the anchor for the (ta) rule. Indeed, $said(P, X)$ satisfies all the conditions of being an anchor, except that the variable P appearing here is not an infon variable. Note that P ranges over the names of the processes. Therefore, for each P, we can create a new $said_P$ (and the corresponding $tdon_P$) infon constructor function, and omit P from the rule. There are finitely many processes; therefore the trust application rule can be seen as a compact way of representing finitely many process-dependent trust application rules.

B EHR Case Study

Medical practices, hospitals and other Care Centers create, collect, and manage *electronic health records* (*EHR*s) for the purposes of treating a particular case and sometimes also for compiling the medical history of a patient. An EHR is a single record of healthcare-related information, which can include information of a wide variety of types, including patient demographics, medical history, medications, emergency contact info and so on. Each record has an implicitly associated access control list, via some indirections which facilitate the understanding for patients and clinicians. The rules defining the access control criteria for an EHR will depend, among others, on the type of EHR. For instance, a highly sensitive record of a treatment for depression might be only available to his treating doctor (and perhaps a few others, on certain conditions), while a record of heart disease can be open to all staff, for the case of an emergency.

The scenario we present here, originally described in [3], concerns the information system of an hospital. The system relies upon a central server, which

stores all data about the hospital's employees and patients. Employees of the hospital can interact with the central server; hence it is important that the central server enforces the rules that regulate access to its data. We consider in particular the procedures undertaken by a doctor to (1) access a patient's health record, and (2) request a holiday leave or return from the holidays. Here, we describe the central server process.

Initial knowledge The server stores the name of the doctors, and also which doctor(s) treats which patient(s).

$\{doctor(D) \mid D \text{ is a doctor}\} \cup \{treating\_doc\_of(D, P) \mid D \text{ is the treating doctor of } P\}$

Central Server's Policy Any treating doctor of a patient is able to read the health record of the patient. Moreover, if doctor D_2 replaces doctor D_1 (e.g. in case D_1 goes on vacation), D_2 can access the health records that D_1 could access.

$$can\_read(D, ehr(P)) \leftarrow treating\_doc\_of(D, P)$$
$$can\_read(D_2, ehr(P)) \leftarrow can\_read(D_1, ehr(P)), replaces(D_2, D_1)$$

Central Server thread: Checking patients' health records Any process D may request access to any patient's health record. The request is granted only if the server can deduce that D has access to P's record.

$$\mathsf{rcv}(sig(D, (\mathsf{ehr\_req}, P)) \blacktriangleright \{\}\{\}^{\neg}$$
$$\{can\_read(D, ehr(P))\}\{\}^{\neg} \blacktriangleright$$
$$\quad \mathsf{snd}(sig(cr, hash(ehr(P))), \{ehr(P)\}_{pk(D)})$$

Here, cr is the Central Server's name.

Central Server thread: Holidays request Any doctor D who is not replacing another doctor may request holiday leaves. Before accepting the request, the server finds another doctor X who has not requested holiday, and is not on holiday. If such a doctor X can be found, the server asks X whether, or not, X would replace D. If yes, the server accepts D's holiday request, and lets X replace D.

$$\mathsf{rcv}(sig(D, \mathsf{holidays\_request})) \blacktriangleright \{req\_holiday(D)\}\{\}^{\neg}$$
$$\{doctor(X)\}\{replaces(D, Y), on\_holiday(X), req\_holiday(X)\}^{\neg} \blacktriangleright$$
$$\quad \mathsf{snd}(sig(cr, (X, \mathsf{replace\_request})))$$
$$\mathsf{rcv}(sig(X, \mathsf{replace\_confirm})) \blacktriangleright \{on\_holiday(D), replaces(X, D)\}\{req\_holiday(D)\}^{\neg}$$
$$\{\}\{\}^{\neg} \blacktriangleright \mathsf{snd}(sig(cr, (D, \mathsf{holidays\_accepted})))$$

We note that in this specification, the attacker can replay the signatures, as they do not contain any freshness guarantees. This would lead the server to let X replace D, while X is on holiday. This attack has been reported in our reachability analysis.

Central Server thread: Back from holidays When a doctor D returns from holidays, he/she reports so to the server. The server retracts the statement that D is on holidays, and moreover revokes the rights of any doctor who has replaced D.

$$\mathsf{rcv}(sig(D, \mathsf{back\_from\_holidays})) \blacktriangleright \{\}\{on\_holiday(D), replaces(X, D)\}^{\neg}$$

Similar to the case above, this thread is susceptible to replay attacks.

C A decision procedure for Reach for caps in DC

In this section we give a detailed description of our procedure for deciding the REACH problem for CAPs in **DC**. The procedure, called *reach*, is shown in Algorithm 1 below.

Algorithm 1 Procedure *reach*

REQUIRES: REACH$\langle \mathsf{cap}, G\rangle$, $\mathsf{cap} = ((\Sigma, \mathcal{V}, \mathcal{P}), \pi_1, \cdots, \pi_\ell, \mathbb{A})$

```
I := interleavings({π_1, ··· , π_ℓ})
for all ι ∈ I do
   (res, σ) := solve(cs(ι, G))
   if res = ⊤ then
      return (⊤,ισ)
return (⊥,∅)
```

We start by giving an informal overview of the procedure *reach*. The algorithm hinges upon the notion of *constraints* (defined in detail in Appendix C.1). Constraints model "symbolic" deduction problems: that is, given a "knowledge" (a set of terms or atoms), a "query" (a term or a literal, i.e. a possibly negated atom) and an inference theory, whether there exists an instantiation of the variables under which the query can be inferred from the knowledge. We refer to such an instantiation of the variables as a *solution* of the constraint; there may be zero or more solutions to a constraint. A solution of a set of constraints is an instantiation of the variables that is a solution of all the constraints in the set simultaneously.

We differentiate and group constraints based on some criteria. Firstly, we differentiate *positive constraints* (i.e. constraints whose query is either a term or a non-negated atom) from *negative constraints* (i.e. constraints whose query is a negated atom). Secondly, we differentiate constraints based on the inference theory they refer to. In the context of CAP specifications falling in the **DC** fragment,

constraints over message terms are always solved under the Dolev-Yao inference theory ($\vdash^{\mathbb{A}}$, cf. § 3.1), whereas all other constraints are relative to theories in **AL** (see § 4); we refer to the former as *attacker constraints*, and to the latter as *policy constraints*. Observe that all attacker constraints are positive, while policy constraints can be both positive and negative. Given a set of constraints CS we will denote the disjoint subsets of attacker, positive policy and negative policy constraints in CS as $att(CS)$, $pos(CS)$ and $neg(CS)$ respectively.

The algorithm *reach* generates a constraint sequence $cs(\iota, G)$ for each interleaving ι of the events of the processes in cap, taking also into account the reachability goal G under analysis; a precise definition of $cs(\iota, G)$ is given in C.2. For each event in ι (resp. reachability goal) a corresponding constraint is added to $cs(\iota, G)$ that models the event (resp. goal) in terms of a symbolic deduction problem. Constraints that model the reception of a message or the evaluation of a non-negated atom are positive; constraints that model the evaluation of a negated atom are negative. In the construction of the constraint sequence $cs(\iota, G)$ also a first step to reflect retraction in the CAP specification is taken: each constraint relative to a process π is coupled with a set of associations between the atoms that have appeared (so far) in negative updates and the atoms in the knowledge of π that can be affected by the retracted atoms; that is, the facts in the knowledge that unify with the retracted atom. We will informally refer to this set at to the *retraction set* of the constraint. The symbolic treatment of retracted atoms is explained in detail in C.6.

Each protocol includes a finite number of threads, each being a finite sequence of events, hence there are finitely many interleavings and finitely many constraint sequences generated for the protocol.

The constraint sequence $cs(\iota, G)$ is then input to a constraint reduction system, called *solve*, shown in Algorithm 2 below.

Algorithm 2 Procedure *solve*

REQUIRES: constraint sequence CS
  $\tau = \emptyset$, $\Delta = \emptyset$
  while CS is not trivial && CS is not most-reduced **do**
    non-deterministically execute one of the following
      • if $att(CS)$ is not trivial then $(CS', \sigma, \delta) = reduceDY(CS)$;
      • if $pos(CS) \neq \emptyset$ then $(CS', \sigma, \delta) = solveALPositive(CS)$;
      • if $neg(CS) \neq \emptyset$ then $(CS', \sigma, \delta) = solveALNegative(CS)$;
    $CS := CS'\sigma$, $\tau := \tau \cup \sigma$, $\Delta := \Delta \cup \delta$
  if CS is trivial **then**
    return (T,τ,Δ)
  else
    return (F,$\emptyset$,$\emptyset$)

The reduction system *solve* tries to reduce the constraint sequence $cs(\iota, G)$ to a *trivial* form, i.e. immediately satisfiable. More precisely, a constraint sequence is trivial if for all constraints $c \in att(CS)$ the query of c is a standalone variable, and $pos(CS) = neg(CS) = \emptyset$. The procedure *solve* reduces the constraint sequence as long as one or more of these conditions are not met: at each iteration of the loop, one of the procedures *reduceDY*, *solveALPositive* and *solveALNegative* is called to reduce $att(CS)$, $pos(CS)$ or $neg(CS)$, respectively. Each of these procedures returns a triple (CS', σ, δ), where CS' is the set of constraints into which CS is reduced, σ is a solution for the subset of CS reduced, and δ is a set of disequalities stemming from the retracted atoms and necessary for the soundness of the reduction.

If $att(CS)$ is not trivial, then *solve* calls the constraint reduction system *reduceDY*, to reduce the attacker constraints in CS. The reduction system *reduceDY* is explained in detail in Appendix C.3.

If $pos(CS) \neq \emptyset$, then the proof search algorithm *solveALPositive* for **AL** theories is used to solve the positive policy constraints in CS. The algorithm *solveALPositive*, for an **AL** theory $T = T_{\rightarrow} \sqcup T_{\leftarrow}$, consists of a cut elimination step that accounts for the rules in $T_{\rightarrow}$ and a backward search algorithm that accounts for the rules of $T_{\leftarrow}$. Intuitively, *solveALPositive* tries to "guess" a proof for the symbolic deduction problem represented by the constraint in input. Furthermore, *solveALPositive* takes a second step in treating facts retraction: while finding a proof for the deduction problem, it uses the constraint's retraction set to identify and store the set of disequalities δ necessary for the proof to hold. These disequalities must be respected at all times, therefore every further instantiation of the variables must comply with the disequalities. We describe *solveALPositive* in detail in Appendix C.4, and elaborate on the treatment of retracted atoms in C.6.

Finally, if $neg(CS) \neq \emptyset$, the procedure *solveALNegative* is used to solve the negative constraints in CS. Procedure *solveALNegative* "complements" all the constraints in $neg(CS)$, i.e. turns each negative constraint into a positive constraint expressing the opposite (positive) symbolic deduction problem; this is done by removing the negation in the query atom. The key idea is that the negative constraint is solvable when its complement has no solutions; cf. [16]. To this end, *solveALNegative* calls procedure *solveALPositive* to solve the complement of the negative constraint, and collects the set of all the solutions returned by *solveALPositive*. The set of solutions is then transformed into an equality logic (i.e. quantifier-free first order logic with equality) formula ϕ; ϕ is such that its models are the instantiations that satisfy the complement constraint. Finally, the negation of ϕ is fed to an equality logic SAT solver (e.g. the algorithm of [5]): the solutions found are instantiations under which the complement can not be satisfied, and those therefore satisfy the negative constraint. A detailed description of *solveALNegative* is given in C.5.

At the end of each iteration, the solution found by the procedure called by the procedure *solve* is applied to the new constraint sequence and stored; moreover, the disequalities returned are also stored to ensure further instantiations

of the variable in CS do not violate them. If *solve* reduces CS to a trivial constraint sequence, then it returns the truth value T, the solution of the constraint sequence τ and the set of disequalities Δ. If instead *solve* reduces CS to a non-trivial set of constraint to which no further reduction can be applied, then *solve* returns the truth value F (and a dummy solution and set of disequalities). Observe that the procedure *solve* considers only one of the (finitely many) possible reduction paths, hence the return value refers only to this reduction path. Procedure *reach* invokes *solve* repeatedly to exhaustively search all possible reduction paths.

In the following sections we describe in detail the subroutines of procedures *reach* and *solve*. In Appendix C.1 we introduce formally the notions needed for the subsequent sections. In Appendices C.2, C.3, C.4 and C.5 we explain, respectively, the generation of the constraint sequence, and the reduction procedures for attacker constraints, positive policy constraints and negative policy constraints. In these four section we will consider specifications where no negative update occurs; that is, for all receive events of the form $\mathsf{rcv}(t) \blacktriangleright u_+u_-$ appearing in the specification $u_- = \emptyset$. The symbolic treatment of retracted atoms and its integration in the procedure *solve* is then explained in Appendix C.6. Finally, in Appendix C.7 we prove the correctness (i.e. termination, soundness and completeness) of Algorithm 1.

C.1 Preliminaries

We define formally here the notions of *constraints*, constraints *solutions* and several others that will be used in the next sections.

Consider a finite Horn theory T. The *ground deduction relation* induced by T is a relation $\vdash^T$ such that $(S, u) \in \vdash^T$ iff u is in the closure of S under theory T (cf. § 3.2). A *positive constraint* over T is of the form $S \Vdash^T u$ where $S \subseteq \mathcal{A}_{\Sigma(\mathcal{V})}$, S is finite, and $u \in \mathcal{A}_{\Sigma(\mathcal{V})}$. Intuitively, such constraint represents a "symbolic" deduction problem, namely the problem of finding an instantiation of the variables in $S \cup \{u\}$ under which u can be proved from S; we call such an instantiation a solution of the constraint. More formally, a solution of $S \Vdash^T u$ is a total substitution function $\sigma : var(S \cup \{u\}) \to \mathcal{T}_{\Sigma(\emptyset)}$ such that $S\sigma \vdash^T u\sigma$.

A *negative constraint* over T is of the form $S \not\Vdash^T u$ where $S \subseteq \mathcal{A}_{\Sigma(\mathcal{V})}$, S is finite, and $u \in \mathcal{A}_{\Sigma(\mathcal{V})}$. A solution of $S \not\Vdash^T u$ is a total substitution function $\sigma : var(S \cup \{u\}) \to \mathcal{T}_{\Sigma(\emptyset)}$ such that $S\sigma \not\vdash^T u\sigma$. As opposed to the solution of a positive constraint, the solution of a negative constraint is an instantiation of the variables in $S \cup \{u\}$ under which it is not possible to prove u from S.

We define the *refinement* partial order $\sqsubseteq$ over the set of all the total mappings $\mathcal{V} \to \mathcal{T}_{\Sigma(\mathcal{V})}$ as: $\sigma \sqsubseteq \sigma'$ iff there exists a mapping σ'' such that $\sigma' = \sigma\sigma''$. Two substitution functions σ and σ' are *equivalent*, denoted $\sigma \equiv \sigma'$, iff $\sigma \sqsubseteq \sigma'$ and $\sigma' \sqsubseteq \sigma$; i.e. they are the same modulo alpha renaming of the variables. We overload the notion of solution of a constraint over a theory T by accounting also for non-ground substitution functions, whose ground refinement are solutions of the constraint. More formally, given a positive constraint $c = S \Vdash u$, a (*symbolic*) solution of c is a total substitution function $\sigma : \mathcal{V} \to \mathcal{T}_{\Sigma(\mathcal{V})}$ such that $S\sigma\sigma' \vdash$

$u\sigma\sigma'$ for any ground total substitution $\sigma' : \mathcal{V} \to \mathcal{T}_{\Sigma(\emptyset)}$. Similarly, for a negative constraint $c = S \not\Vdash u$, a (symbolic) solution of c is a total substitution function $\sigma : \mathcal{V} \to \mathcal{T}_{\Sigma(\mathcal{V})}$ such that $S\sigma\sigma' \not\vdash u\sigma\sigma'$ for any ground total substitution $\sigma' : \mathcal{V} \to \mathcal{T}_{\Sigma(\emptyset)}$. Intuitively, a symbolic solution for a constraint instantiates only the variables that are necessary to solve the constraint, without binding the remaining ones.

A *constraint system* over T is a finite set of constraints over T. A constraint system C over a finite set of finite Horn theories $\{T_1, \cdots, T_\ell\}$ is the union of finitely many constraint systems over any of the theories $T_1, \cdots, T_\ell$. A *solution* for C is a total substitution function σ that solves all the constraints in C simultaneously.

Given a CAP specification cap, we can partition the variables occurring in cap into two sets $\mathcal{V}_a$ and $\mathcal{V}_g$ of, respectively, *attacker originated* variables (i.e. appearing for the first time in cap in a receive event) and *guard originated* variables (i.e. appearing for the first time in cap in a guard event). Observe that $\mathcal{V}_a \cap \mathcal{V}_g = \emptyset$ and $\mathcal{V}_a \cup \mathcal{V}_g \subseteq \mathcal{V}_{\mathsf{msg}}$, that is, $\mathcal{V}_a$ and $\mathcal{V}_g$ are disjoint and their members are of type msg.

A constraint $T \Vdash u$ is *simple* if $u \in \mathcal{V}$ and u is not a subterm of any term in T. Intuitively this means that we can instantiate u with any value that follows from T. Observe that a policy constraint $T' \Vdash u'$ can never be simple, because u' ranges over terms of type infon. A constraint system C is trivial iff all constraints in C are simple.

To avoid cluttering, in the following we will write T-*constraint* (resp. $T_1, \cdots, T_n$-*constraint system*) as a synonym of (either positive or negative) constraint over T (resp. constraint system over $T_1, \cdots, T_n$).

C.2 Generating the constraint systems

Let $\mathsf{cap} = ((\Sigma, \mathcal{V}, \mathcal{P}), \pi_1, \cdots, \pi_\ell, \mathbb{A})$ be a CAP in **DC**, with $\pi_i = (\eta_i, \Omega_i, \mathbf{I}_i)$ for $i \in [1, \ell]$. A *symbolic* (i.e. non-ground) configuration of cap is a tuple

$$((\eta_1, L_1, \mathbf{I}_1), \cdots, (\eta_\ell, L_\ell, \mathbf{I}_\ell), (\Omega_{\mathbb{A}}, \mathbf{I}_{\mathbb{A}})),$$

where $\eta_1, \cdots, \eta_\ell$ are finite sets of threads, $\Omega_1, \cdots, \Omega_\ell, \Omega_{\mathbb{A}}$ are finite sets of atoms and $\mathbf{I}_1, \cdots, \mathbf{I}_\ell, \mathbf{I}_{\mathbb{A}}$ are finite sets of Horn clauses. A symbolic configuration represents a state in an abstract execution of the CAP . With respect to concrete configurations (see § 3.2) symbolic configurations might contain variables; they can represent a (possibly infinite) set of concrete configurations, all those obtainable by grounding the variables in the symbolic configuration. In the following we omit the policies $\mathbf{I}_1, \cdots, \mathbf{I}_\ell$ of the processes and the ground deduction relation of the attacker, when they are clear from the context.

Let Z be the set of all symbolic configurations of cap, i.e. the set of all tuples of the form $z = (\pi_1, \cdots, \pi_\ell, \mathbb{A})$. We define the relation $\rightarrowtail \subseteq Z \times E \times Z$ as: $(z, e, z') \in \rightarrowtail$ iff $z = (\pi_1, \cdots, \pi_i, \cdots, \pi_\ell, \mathbb{A})$, $z' = (\pi_1, \cdots, \pi'_i, \cdots, \pi_\ell, \mathbb{A}')$, $\mathbb{A} = (\Omega_{\mathbb{A}}, \vdash^{\mathbb{A}})$, $\pi_i = (\eta, \Omega, \mathbf{I})$ and $p = e \cdot p' \in \eta$ for some thread p' and $1 \leq i \leq \ell$, and one of the following conditions hold:

– $e = g \blacktriangleright \mathsf{snd}(t)$, $\mathbb{A}' = (\Omega_{\mathbb{A}} \cup \{t\}, \vdash^{\mathbb{A}})$ and $\pi_i' = (\eta \setminus \{p\} \cup \{p'\}, \Omega, \mathbf{I})$;
– $e = \mathsf{rcv}(t) \blacktriangleright (u_+, u_-)$, $\mathbb{A}' = \mathbb{A}$ and $\pi_i' = (\eta \setminus \{p\} \cup \{p'\}, \Omega \cup u_+, \mathbf{I})$. [4]

We write $z \overset{e}{\rightarrowtail} z'$ for $(z, e, z') \in \rightarrowtail$.

An *interleaving* of cap is an alternating sequence of symbolic configurations and events $\iota = z_0 e_1 z_1 \cdots z_{n-1} e_n z_n$ such that z_0 is the initial configuration of cap and, for all $i \in [1, n]$, $z_{i-1} \overset{e_i}{\rightarrowtail} z_i$. As opposed to traces, in an interleaving the variables introduced by an event are not instantiated, but carried over in the following symbolic configurations, thus symbolically representing a (possibly infinite) set of concrete traces. Also differently from traces, interleavings are finite; this is implied by the fact that CAP specifications contain finitely many threads, each of finite length, and thus induce finitely many interleavings.

A substitution $\sigma : \mathcal{V} \to \mathcal{T}_{\Sigma(\emptyset)}$ *realizes* an interleaving ι if $\iota\sigma$ is a trace of the CAP . We say that an interleaving ι is realizable iff there exists at least one substitution σ that realizes ι.

Let $\iota = z_0 e_1 \cdots z_{i-1} e_i z_i \cdots e_n z_n$ be an interleaving of cap. To simplify the presentation, in the remainder of the paper we assume that for all events e_i in ι (with $i \in [1, n]$), if $e_i = (g_\exists, g_\nexists) \blacktriangleright \mathsf{snd}(m)$ then either $g_\exists = \{a\}$ and $g_\nexists = \emptyset$, or $g_\exists = \emptyset$ and $g_\nexists = \{a\}$; that is, only one of $g_\exists$ and $g_\nexists$ is non-empty, and it is a singleton set. Similarly, we will assume goals of the form $G = \pi : f$, for $\pi \in \{\pi_1, \cdots, \pi_\ell\}$ and $f \in \mathcal{A}_{\Sigma(\emptyset)}$, or $G = \mathbb{A} : m$, for $m \in \mathcal{T}_{\Sigma(\emptyset)}$. It will be apparent in the construction of the constraint system how guards and goals containing more than one atom can be represented using additional constraints.

Below, we describe how $cs(\iota, G)$ is generated for interleaving ι and goal G. Intuitively, each event of the interleaving ι gives rise to one constraint in the constraint system $cs(\iota)$. Let e be a guarded send $e = (g_\exists, g_\nexists) \blacktriangleright \mathsf{snd}(t)$, executed at configuration z by thread p belonging to process π. If $g_\exists = \{a\}$ then the π-constraint $\Omega_\pi \Vdash^\pi a$ is added to the constraint system, checking that atom a evaluates to true against the policy of π at configuration z. If $g_\nexists = \{a\}$ then a negative π-constraint $\Omega_\pi \nVdash^\pi a$ is added to the constraint system, checking that the atom a evaluates to false against the policy of π at configuration z. We refer to such constraints as (resp. positive and negative) *policy* constraints, since they ascertain the satisfaction of policy requirements. because the For a receive event $e = \mathsf{rcv}(t) \blacktriangleright u$, executed at configuration z, the $\mathbb{A}$-constraint $\Omega_{\mathbb{A}} \Vdash^{\mathbb{A}} \mathcal{K}(t)$ is added to the constraint system, checking that the term t can be generated from the attacker's knowledge set $\Omega_{\mathbb{A}}$ at configuration z. We refer to such constraints as *attacker* constraints, as they model the messages that the attacker should generate. Notice that the attacker constraints in $cs(\iota)$ are always positive.

Additionally, one constraint is added to the constraint system to model the goal G. If $G = \pi : f$, with $\pi \in \{\pi_1, \cdots, \pi_\ell\}$ and $f \in \mathcal{A}_{\Sigma(\emptyset)}$ we add the policy constraint $\Omega_\pi \Vdash^\pi f$, where Ω_π is the extentional knowledge of process π at configuration z_n (i.e. the configuration reached after full execution of the inter-

[4] Negative updates are here deliberately ignored; their treatment will be explained separately later in Appendix /refapp:retr.

leaving). If $G = \mathbb{A} : m$ with $m \in \mathcal{T}_{\Sigma(\emptyset)}$ we add the attacker constraint $\Omega_{\mathbb{A}} \Vdash^{\mathbb{A}} m$, where $\Omega_{\mathbb{A}}$ is the knowledge set of the attacker at configuration z_n. [5]

Formally, for an interleaving $\iota = z_0 e_1 z_1 \cdots z_{n-1}$ of cap and a goal G, the constraint system $cs(\iota, G)$ generated is $cs(\iota) \cup cs(G)$, that is, the union of the constraints generated to model the interleaving ($cs(\iota)$) and the constraints generated to model the goal ($cs(G)$); both are defined below:

$$cs(\iota) = \begin{cases} \emptyset & \text{if } \iota = \epsilon \quad \text{(empty interleaving)} \\ \{\Omega_\pi \Vdash^\pi g\} \cup cs(\iota') & \text{if } \iota = ze\iota', e = (\{g\}, \emptyset) \blacktriangleright \mathsf{snd}(t), e \text{ is an event of process } \pi \\ \{\Omega_\pi \nVdash^\pi g\} \cup cs(\iota') & \text{if } \iota = ze\iota', e = (\emptyset, \{g\}) \blacktriangleright \mathsf{snd}(t), e \text{ is an event of process } \pi \\ \{\Omega_{\mathbb{A}} \Vdash^{\mathbb{A}} t\} \cup cs(\iota') & \text{if } \iota = ze\iota' \text{ and } e = \mathsf{rcv}(t) \blacktriangleright u \end{cases}$$

$$cs(G) = \begin{cases} \{\Omega_\pi \Vdash^\pi f\} & \text{if } G = \pi : f \\ \{\Omega_{\mathbb{A}} \Vdash^{\mathbb{A}} m\} & \text{if } G = \mathbb{A} : m \end{cases}$$

where Ω_i denotes the symbolic extensional knowledge of $i \in \{\pi_1, \cdots, \pi_\ell, \mathbb{A}\}$ at configuration z. Since each CAP results in a finite number of interleavings, there are finitely many constraint systems generated for each CAP .

The constraint system $cs(\iota, G)$ is input to *solve*, shown in Algorithm 2. Solve tries to reduce $cs(\iota, G)$ into a trivial constraint system, by invoking the reduction procedures *reduceDY*, *solveALPositive* and *solveALNegative*. In the following sections, we discuss in detail these procedures.

C.3 Solving attacker constraints

The set $att(cs(\iota, G))$ of attacker constraints in $cs(\iota, G)$ is solved by the procedure *reduceDY*, shown in Figure 2. *reduceDY* is an adaptation of the nondeterministic constraint reduction system formerly presented in [15] to decide the problem of reachability in security protocols under the usual Dolev-Yao attacker theory, shown in table 1 below.

Composition Rules	**Decomposition Rules**
$\mathcal{K}((X,Y)) \leftarrow \mathcal{K}(X), \mathcal{K}(Y)$	$\mathcal{K}(X) \leftarrow \mathcal{K}((X,Y))$
$\mathcal{K}(sig(X,Y)) \leftarrow \mathcal{K}(X), \mathcal{K}(Y)$	$\mathcal{K}(Y) \leftarrow \mathcal{K}((X,Y))$
$\mathcal{K}(\{\!|X|\!\}_Y) \leftarrow \mathcal{K}(X), \mathcal{K}(Y)$	$\mathcal{K}(X) \leftarrow \mathcal{K}(\{\!|X|\!\}_Y), \mathcal{K}(Y)$
$\mathcal{K}(\{X\}_Y) \leftarrow \mathcal{K}(X), \mathcal{K}(Y)$	$\mathcal{K}(X) \leftarrow \mathcal{K}(\{X\}_{\mathrm{pk}(Y)}), \mathcal{K}(\mathrm{sk}(Y))$
$\mathcal{K}(h(X)) \leftarrow \mathcal{K}(X)$	

Table 1. Message inference capabilities of the Dolev-Yao attacker

[5] We remark that even though we do not consider partial executions here (e.g. as in [16]) it is easy to account for them: one possibility is to consider variants of the constraint system generated where the constraint $\Omega \Vdash^\pi u$ that models the goal is replaced with a constraint $\Omega' \Vdash^\pi u$, where Ω' is the extentional knowledge (or knowledge set, if $\pi = \mathbb{A}$) of π at configuration z_i, with $i \in [0, n]$.

Intuitively, *reduceDY* non-deterministically applies the rules of Figure 2 repeatedly on a constraint system CS containing only attacker constraints until a *most reduced* constraint system CS' is reached. A constraint system CS' is most-reduced iff no reduction rule is applicable to CS'; that is, $\nexists \tau,\, CS''.\ CS' \rightsquigarrow_\tau CS''$. A most-reduced constraint system is *trivial* iff all its constraints are of the form $T \Vdash^{\mathbb{A}} X$, with $X \in \mathcal{V}$. A trivial constraint system can be immediately solved. If CS' is trivial then CS is solvable. However, if CS' is not trivial, then *reduceDY* backtracks, i.e. it tries to apply on CS another sequence of the rules of Figure 2. There are finitely many possible reduction paths.

$$
\begin{array}{lll}
(R_1)\ C \cup \{T \Vdash^{\mathbb{A}} u\} \rightsquigarrow\ C & & \text{if } T \cup \{x \mid T' \Vdash^{\mathbb{A}} x \in C, T' \subset T\} \vdash^{\mathbb{A}} u \\
(D_1)\ C \cup \{T \Vdash^{\mathbb{A}} u\} \rightsquigarrow\ C \cup \{T \Vdash^{\mathbb{A}} t, T \Vdash^{\mathbb{A}} v\} & & \text{if } u = f(t,v) \text{ and } f \in \{(\cdot,\cdot), \{\cdot\}., \{\!|\cdot|\!\}., sig(\cdot,\cdot)\} \\
(D_2)\ C \cup \{T \Vdash^{\mathbb{A}} u\} \rightsquigarrow\ C \cup \{T \Vdash^{\mathbb{A}} v\} & & \text{if } u = h(v) \\
(U_1)\ C \cup \{T \Vdash^{\mathbb{A}} u\} \rightsquigarrow_\tau C\tau \cup \{T\tau \Vdash^{\mathbb{A}} u\tau\} & & \text{if } \tau = mgu(u,t), t \in sub(T), t \neq u, t, u \notin \mathcal{V}_a \\
(U_2)\ C \cup \{T \Vdash^{\mathbb{A}} u\} \rightsquigarrow_\tau C\tau \cup \{T\tau \Vdash^{\mathbb{A}} u\tau\} & & \text{if } \tau = mgu(t,v), t, v \in sub(T), t \neq v,\ t, v \notin \mathcal{V}_a \\
(U_3)\ C \cup \{T \Vdash^{\mathbb{A}} u\} \rightsquigarrow_\tau C\tau \cup \{T\tau \Vdash^{\mathbb{A}} u\tau\} & & \text{if } \tau = mgu(t,v), \{w\}_{\mathrm{pk}(t)}, \\
 & & \quad \mathrm{sk}(v) \in sub(T), t \neq v, t \in \mathcal{V}_a \vee v \in \mathcal{V}_a
\end{array}
$$

Fig. 2. Constraint reduction system *reduceDY*

Below, we briefly explain each of the reduction rules of Figure 2. Rule R_1 removes a constraint $T \Vdash^{\mathbb{A}} u$. It assumes a decision procedure for the DY ground deduction problem $\vdash^{\mathbb{A}}$, reflecting the DY attacker capabilities (see table 1). It is well-known that $\vdash^{\mathbb{A}}$ is decidable, and it also meets Tarski's conditions for consequence relations. The constraint is removed if $T \cup \{x \mid T' \Vdash^{\mathbb{A}} x \in C, T' \subset T\} \vdash^{\mathbb{A}} u$. Rules D_1 and D_2 decompose an attacker constraint $T \Vdash^{\mathbb{A}} u$ into simpler constraints, when the most external constructor used in u is a function available to the attacker. Rules U_1 and U_2, given an attacker constraint $T \Vdash^{\mathbb{A}} u$, unify some subterm of T with, respectively, u or another subterm of T. Finally, rule U_3 unifies two subterms of T, where the subterms represent a secret key known to the attacker and a corresponding public key used for asymmetric encryption.

In the following sections we prove the termination and correctness of the reduction procedure *reduceDY*. A proof Π of $S \vdash a$ over domain D, with $S \subseteq D$, S finite, $a \in D$ and $\vdash$: $2^D \times D$ being a ground deduction relation, is called *normal* if no label $S' \vdash a'$ appears more than once on the same branch of Π. Given a sequence $T_1 \subseteq T_2 \cdots T_{n-1} \subseteq T_n$ (totally ordered w.r.t. set inclusion) of finite subsets of the domain D, we say that a proof Π of $T_i \vdash a$, with $1 \leq i \leq n$, is *left-minimal* if Π does not hold when substituting T_i with T_j, for any $1 \leq j < i$, in all labels of Π. We recall Tarski's conditions for consequence relations, which hold in particular for $\vdash^{\mathbb{A}}$:

1. *Set membership*: if $u \in S$, then $S \vdash^T u$;
2. *Monotonicity*: if $S \vdash^T u$ and $S \subseteq S'$, then $S' \vdash^T u$;

3. *Consequence*: if $S \vdash^T u$ and $S \cup \{u\} \vdash^T v$, then $S \vdash^T v$.

Observe also that the constraint system $cs(\iota, G)$ has the following properties:

- *monotonicity.* Let $cs_{\mathbb{A}}(\iota, G) = \{T_1 \Vdash^{\mathbb{A}} u_1, \cdots, T_\ell \Vdash^{\mathbb{A}} u_\ell\}$. The term sets $T_1, \cdots, T_\ell$ are totally ordered by set inclusion.
- *attacker origination.* Let $T \Vdash^{\mathbb{A}} u$ be a constraint in $cs(\iota, G)$, and $v \in var(T) \cap \mathcal{V}_a$. Then there exists $T_x \Vdash^{\mathbb{A}} u_x \in cs_{\mathbb{A}}(\iota)$ such that $T_x = min(\{T' \mid T' \Vdash^{\mathbb{A}} u' \in cs(\iota), x \in var(u')\})$ and $T_x \subset T$.

Termination

Theorem 2. *The procedure reduceDY terminates for all inputs generated from* CAP *specifications in the* **DC** *fragment.*

Proof. Associate to any positive constraint system C a pair of non-negative integers $\nu(C) = (n, m)$ where n is the number of variables appearing in C and m is the number of function applications and constants in the right hand sides of the constraints in C (also counting repetitions). Consider a lexicographic ordering on pairs $\leq_{lex}$. We show that, for any two constraint systems C and C', $C \rightsquigarrow C'$ implies $\nu(C) >_{lex} \nu(C')$. In fact, rules U_1, U_2 and U_3 decrease n strictly. All other rules, while not increasing n, decrease m strictly.

Soundness

Theorem 3. *The procedure reduceDY is sound. That is, for two positive constraint systems C and C' such that $C \rightsquigarrow_\tau C'$ and a solution ρ of C', $\tau\rho$ is a solution of C.*

Proof. Let C and C' be two constraint systems such that $C \rightsquigarrow_\tau C'$, and ρ be a solution of C'. We show that $\tau\rho$ is then a solution of C.

We condition on the rule applied:

- (R_1). Let $T \Vdash^{\mathbb{A}} u$ be the constraint eliminated from C. Hence, $C' = C \setminus T \Vdash^{\mathbb{A}} u$. As in the case of R_1, ρ is a solution for all constraints $(T' \Vdash^{\pi} u') \neq (T \Vdash^{\mathbb{A}} u)$ in C. Hence we need to show that $T\rho \vdash^{\mathbb{A}} u\rho$. Recall that $T \cup \{x \mid T' \Vdash^{\mathbb{A}} x \in C, T' \subset T\} \vdash^{\mathbb{A}} u$, and consequently also $T\rho \cup \{x\rho \mid T' \Vdash^{\mathbb{A}} x, T' \subset T\} \vdash^{\mathbb{A}} u\rho$. By Tarski's monotonicity property, $T'\rho \vdash^{\mathbb{A}} x\rho$ (with $T' \subset T$) implies $T\rho \vdash^{\mathbb{A}} x\rho$. Remark that this is the case for all constraints in C whose left hand side is smaller than T. Then, by Tarski's consequence property, $T\rho \cup \{x\rho \mid T' \Vdash^{\mathbb{A}} x, T' \subset T\} \vdash^{\mathbb{A}} u\rho$ and $T\rho \vdash^{\mathbb{A}} x\rho$ (for all $T' \Vdash^{\mathbb{A}} x$ in C with $T' \subset T$), from which follows $T\rho \vdash u\rho$.
- (D_1) and (D_2). Let $T \Vdash^{\mathbb{A}} f(u, v)$ be the constraint in C, replaced in C' by $T \Vdash^{\mathbb{A}} u$ and $T \Vdash^{\mathbb{A}} v$. Since $T\rho \vdash^{\mathbb{A}} u\rho$ and $T\rho \vdash^{\mathbb{A}} v\rho$, $T\rho \vdash^{\pi} u\rho$ follows from the corresponding inference rule in the attacker model (cf. table 1).
- (U_1), (U_2) and (U_3). For any constraint $T \Vdash^{\mathbb{A}} u \in C$ there is a constraint $T\tau \Vdash^{\mathbb{A}} u\tau \in C'$. Since ρ is a solution of C', $(T\tau)\rho \vdash^{\pi} (u\tau)\rho$, that is $T(\tau\rho) \vdash^{\pi} u(\tau\rho)$.

Completeness

Theorem 4. *The procedure reduceDY is complete. That is, for a positive constraint system C that is not most-reduced and a solution ω of C, there exist a constraint system C' and a solution ρ of C' such that $C \rightsquigarrow_\tau C'$ and $\omega = \tau\rho$.*

Proof. Let C be a constraint system that is not most-reduced, and ω be a solution of C. Then, there exists a constraint $T \Vdash^{\mathbb{A}} u \in C$ whose right hand side is a non-variable term. Let $T \Vdash^{\mathbb{A}} u$ be such that for all $T' \Vdash^{\mathbb{A}} u' \in C$, with $T' \subset T$, $u' \in \mathcal{V}$. Since ω is a solution of C, then $T\omega \vdash^{\mathbb{A}} u\omega$. Let Π be a normal left-minimal proof of $T\omega \vdash^{\mathbb{A}} u\omega$. We reason inductively on the structure of Π, conditioning on the last rule applied. We consider the following cases:

- **Composition.** If the last rule applied in Π is a composition rule (cf. table 1), then u is of the form $u = f(x, y)$, with $f \in \{(\cdot,\cdot), \{\!|\cdot|\!\}_\cdot, \{\cdot\}_\cdot, sig(\cdot,\cdot)\}$ then from $T\omega \vdash^{\mathbb{A}} u\omega$ we derive $T\omega \vdash^{\mathbb{A}} x\omega$ and $T\omega \vdash^{\mathbb{A}} y\omega$ (similarly if $u = h(x)$). Then the rule D_1 (or, respectively, D_2) is applicable with $\tau = \emptyset$, resulting in removing the constraint $T \Vdash^{\mathbb{A}} u$ and introducing the constraints $T \Vdash^{\mathbb{A}} x$ and $T \Vdash^{\mathbb{A}} y$. We need to show that ω is a solution for the newly introduced constraints. This is immediate, because in Π we have $T\omega \vdash^{\mathbb{A}} x\omega$ and $T\omega \vdash^{\mathbb{A}} y\omega$.
- **Decomposition or axiom.** If the last rule applied in Π is the axiom or a decomposition rule (cf. table 1), then due to Lemma 1 there exists $t \in sub(T)$, $t \notin \mathcal{V}_a$, such that $t\omega = u\omega$.
  If $t \neq u$ then rule U_1 is applicable. Since $\tau = mgu(u, t)$, by definition of mgu for any substitution ω such that $t\omega = u\omega$ there exists a substitution ρ such that $\omega = \tau\rho$.
  If $t = u$, then $u \in sub(T)$.
  If there exists $t' \in sub(T)$ such $t' \notin \mathcal{V}_a$ and $t' \neq t$, then rule U_2 is applicable. Since $\tau = mgu(t, t')$, by definition of mgu there exists ρ that solves C' and such that $\omega = \tau\rho$.
  If there exist $\{t_1\}_{\mathrm{pk}(t_2)}, \mathrm{sk}(t_3) \in sub(T)$, with either t_2 or t_3 being an attacker originated variable, then rule U_3 is applicable and again by definition of mgu there exists ρ that solves C' and such that $\omega = \tau\rho$.
  If none of the above applies, then the conditions for lemma 2 apply, hence $T \cup \{x \mid (T' \Vdash^{\mathbb{A}} x) \in C, T' \subset T\} \vdash^{\mathbb{A}} u$. Consequently, rule R_1 is applicable.

Lemma 1. *Let C be a constraint system that is not most-reduced, ω be a solution of C, and T be a set of facts such that for all constraints $T' \Vdash^{\mathbb{A}} x \in C$, with $T' \subset T$, $x \in \mathcal{V}$. If there exists a normal left-minimal proof of $T\omega \vdash u\omega$ whose last rule is a decomposition rule or the axiom rule, then there exists $t \in sub(T)$ such that $t\omega = u\omega$ and $t \notin \mathcal{V}_a$.*

Proof. It is obvious that if the last rule in a normal left-minimal proof of $T\omega \vdash u\omega$ is a decomposition rule then there exists $t \in sub(T)$ such that $t\omega = u\omega$. It remains to prove that $t \notin \mathcal{V}_a$. By contradiction, assume $t \in \mathcal{V}_a$. Then, by the attacker origination property (cf. Appendix C.2), there exists $T_t \Vdash^{\mathbb{A}} t \in C$ such that $t \notin sub(T_t)$ and $T_t \subset T$. But then $T_t\omega \vdash^{\mathbb{A}} t\omega = u\omega$, which contradicts the left-minimality of $T\omega \vdash^{\mathbb{A}} u\omega$.

In the following we write $pv(C, T)$, for a constraint system C and a finite set of terms T, to denote the set $\{x \mid (T' \Vdash^{\mathbb{A}} x) \in C, T' \subset T\}$. Also, given a set of terms T, a term u and a ground deduction relation $\vdash$ we write $T \vdash u$ to denote the ground deduction problem where variables in T and in u are considered as constants. Lemma 2 has been proved in [15]. We give a slightly modified (and simpler) proof here, that also accounts for the less restrictive property of attacker origination.

Lemma 2. *Let C be a constraint system, ω be a solution of C, u be a term and T be the left hand side of a constraint in C, such that:*

1. *for any constraint $T' \vdash^{\mathbb{A}} x$ in C, with $T' \subset T$, x is a variable*
2. *there are no two distinct terms $t_1, t_2 \in sub(T)$ such that $t_1, t_2 \notin \mathcal{V}_a$ and $t_1\omega = t_2\omega$*
3. *there are no two terms $\{t_1\}_{\mathsf{pk}(t_2)}, \mathsf{sk}(t_3) \in T$ such that either $t_2 \in \mathcal{V}_a$ or $t_3 \in \mathcal{V}_a$ and $t_2 \neq t_3$*
4. *u is a subterm of T such that $u \notin \mathcal{V}_a$*
5. *$T\omega \vdash^{\mathbb{A}} u\omega$*

Then $T \cup pv(C, T) \vdash^{\mathbb{A}} u$.

Proof. Firstly, we remark that due to condition (2) no guard originated variable appears in T (and, as a consequence of (4), neither in u); that is $var(T \cup \{u\}) \cap \mathcal{V}_g = \emptyset$. In absence of ambiguity, in the remainder of the proof we will refer simply to variables instead of attacker originated variables.

Consider a normal left-minimal proof Π of $T\omega \vdash^{\mathbb{A}} u\omega$. Without loss of generality we consider T as the minimal left hand side of the constraints in C for Π to be left-minimal (alternatively T can be replaced in Π with the minimal T' for which Π still holds). We prove the lemma by induction on the depth of Π, conditioning on the nature of the last rule applied in Π.

Axiom or decomposition rule. If Π ends with a decomposition rule or an axiom, then there is a term t such that $T\omega \vdash^{\mathbb{A}} t\omega$ and possibly another term v (again such that $T\omega \vdash^{\mathbb{A}} v\omega$) that are necessary for the application of the last rule in Π. Namely,

- $t = t'$, if the last rule applied is an axiom
- $t = (t', t'')$ or $t = (t'', t')$, if the last rule applied is unpairing
- $t = \{t'\}_{\mathrm{pk}(v')}$ and $v = \mathrm{sk}(v'')$, with $v'\omega = v''\omega$, if the last rule applied is asymmetric decryption
- $t = \{\!|t'|\!\}_{v'}$ and $v = v''$, with $v'\omega = v''\omega$, if the last rule applied is symmetric decryption

where $t'\omega = u\omega$. Due to normality, t can not be the result of a compositional rule in Π, hence $t \in sub(T)$.

Assume, by contradiction, that $t' \in \mathcal{V}$. Then by attacker origination (cf. Appendix C.2) there exists $T_{t'} \Vdash^{\mathbb{A}} t'$ in C, with $T_{t'} \subset T$, of which ω is a solution. But then $T_{t'}\omega \vdash^{\mathbb{A}} t'\omega = u\omega$, contradicting the left-minimality of Π. Hence t' can

not be a variable. On the other hand, due to conditions (2) and (4), t' can not be a non-variable term distinct from u. This implies that $t' = u$. In the case of an axiom or of unpairing, we can then conclude that $T \cup pv(C,T) \vdash^{\mathbb{A}} u$.

In the case of asymmetric decryption also term $v = \mathrm{sk}(v'')$ is needed. Since $\mathrm{sk}(v'')$ can not be the result of the application of a compositional rule in Π, it must be a subterm of T. Due to condition (3) there can not exist $t = \{t'\}_{\mathrm{pk}(v')}$ and $v = \mathrm{sk}(v'')$ with at least one of v' and v'' being a variable distinct from the other. It follows that $v' = v''$, and therefore that the asymmetric decryption rule is applicable to t and v thus obtaining u. Hence, $T \cup pv(C,T) \vdash^{\mathbb{A}} u$.

In the case of symmetric decryption also term v'' is needed (the key used for encryption in t), with $T\omega \vdash^{\mathbb{A}} v''\omega$. Due to condition (2) either $v' = v''$, or one of the two is a variable. In the former case, the claim of the lemma is immediate. We first consider the case v' is a variable, and then turn to the case v' is not a variable (thus v'' must be a variable).

- If v' is a variable then, by the attacker origination property, there exists $T_{v'} \Vdash^{\mathbb{A}} v'$ in the constraint system C where $T_{v'} \subset T$. Therefore, $T \cup pv(C,T) \vdash^{\mathbb{A}} v'$. Then by applying the symmetric decryption rule we conclude that $T \cup pv(C,T) \vdash^{\mathbb{A}} u$.
- Now, suppose v'' is a variable and v' is not a variable. By the proof Π of $T\omega \vdash^{\mathbb{A}} u\omega$ we have $T\omega \vdash^{\mathbb{A}} v''\omega$. Then, $v'\omega = v''\omega$ implies $T\omega \vdash^{\mathbb{A}} v'\omega$. We remark that v' is not a variable. Therefore, by the induction hypothesis, we have $T \cup pv(C,T) \vdash^{\mathbb{A}} v'$. Then applying the symmetric decryption rule yields $T \cup pv(C,T) \vdash^{\mathbb{A}} u$.

Composition rule. If Π ends with a composition rule, then u (being a non-variable term) has a definite form. Namely,

- $u = (v_1, v_2)$, if the last rule applied is pairing
- $u = h(v_1)$, if the last rule applied is hashing
- $u = \{v_1\}_{v_2}$, if the last rule applied is asymmetric encryption
- $u = \{\!|v_1|\!\}_{v_2}$, if the last rule applied is symmetric encryption

Consider any term v needed to construct u. If $v \in \mathcal{V}_a$ then v appears in T due to condition (4). Then, by attacker origination, there exists $T_v \Vdash^{\mathbb{A}} v$ in C, with $T_v \subset T$, hence $T \cup pv(C,T) \vdash^{\mathbb{A}} v$. If v is not a variable, then by induction hypothesis $T \cup pv(C,T) \vdash^{\mathbb{A}} v$. Therefore, applying the corresponding composition rule, $T \cup pv(C,T) \vdash^{\mathbb{A}} u$.

C.4 Solving positive policy constraints

In this section, we present a proof search procedure for **AL** theories. The procedure, called *ps* (standing for *proof search*), is shown in Figure 3. The procedure *ps* is the "heart" of the procedure *solveALPositive*: the latter calls repeatedly *ps* for each constraint in $pos(cs(\iota, G))$, to find a solution τ that solves all of them simultaneously.

Consider an **AL** theory $T = T_{\leftarrow} \sqcup T_{\rightarrow}$, and write $\vdash$ for the ground deduction relation induced by T. As a convention, we write rules in $T_{\rightarrow}$ as $t \leftarrow t_1, \cdots, t_n, a$ so that the anchor is the last premise (in this case, a) of the rule. The procedure ps is a non-deterministic constraint reduction system. The input to ps is a finite set of T-constraints; typically, a singleton T-constraint system $C_0 = \{\Omega \Vdash u\}$. Intuitively, ps "guesses" tentative proof trees for $\Omega\tau \vdash u\tau$, with τ being a solution of $\Omega \Vdash u$. The constraint system created and further reduced by ps represents the set of (not yet discharged) leaves for the tentative proof tree. If ps can discharge all the leaves, then ps outputs the solution τ. However, if ps fails to discharge all the leaves of the tentative proof, then ps backtracks. There are finitely many guesses that ps can make (this is stated by Theorem 5 below). If ps fails to discharge the leaves (i.e. assumptions) for all the (guessed) tentative proofs, then ps returns a failure, indicating $\neg\exists\tau.\ \Omega\tau \vdash u\tau$.

$$(1)\ C \cup \{S \sqcup \{t\} \Vdash u\} \rightsquigarrow_\tau C\tau \quad \text{if } \tau = mgu(u,t)$$

$$(2)\ C \cup \{S \sqcup \{z\} \Vdash u\} \rightsquigarrow_\tau C \cup \{(S \Vdash t_1, \cdots, S \Vdash t_n, S \cup \{t\} \Vdash u)\tau\}$$
$$\text{if } \tau = mgu(z,a) \text{ for some } t \leftarrow t_1, \cdots, t_n, a \in T_{\rightarrow}$$

$$(3)\qquad C \cup \{S \Vdash u\} \rightsquigarrow_\tau (C \cup \{S \Vdash t_1, \cdots, S \Vdash t_n\})\tau$$
$$\text{if } \tau = mgu(u,t) \text{ for some } t \leftarrow t_1, \cdots, t_n \in T_{\leftarrow}$$

Fig. 3. Constraint reduction system for **AL** proof search

The procedure ps applies non-deterministically one of the constraint reduction rules (shown in Figure 3) as long as the constraint system is not empty. The reduction rules reflect the Horn clauses in T and the axiom rule. We write $C \rightsquigarrow_\tau C'$ to denote that the constraint system C is reduced to the constraint system C' by applying the partial substitution $\tau : var(C) \to \mathcal{T}_{\Sigma(\mathcal{V})}$.

- Rule (1) discharges one of the leaves using unification. For a constraint $S \sqcup \{t\} \Vdash u \in C$, if u can be unified with a term t in the left hand side of the constraint, then the constraint is removed, and their most general unifier τ is applied to the remaining (if any) constraints in C.
- Rule (2) applies a rule $t \leftarrow t_1, \cdots, t_n, a \in T_{\rightarrow}$ to a constraint $(S \sqcup \{z\} \Vdash u) \in C$. The intuition behind the rule lies in the fact that the anchor a can only be found in the extentional knowledge or derived with another application of R. If z unifies with a, and τ is the most general unifier of z and a, then the constraint is replaced by $n+1$ new constraints: the first n constraints, $(S \Vdash t_1, \cdots, S \Vdash t_n)\tau$, check that the premises of the rule (except for a) can be proved from S; the last constraint, $(S \cup \{t\} \Vdash u)\tau$, checks whether u can be proved knowing S and t. Notice that z does no longer appear in the left hand side of the new constraints: this ensures termination of ps, and does not compromise its completeness (cf. Theorems 5 and 6).

– Finally, rule (3) applies a rule $t \leftarrow t_1, \cdots, t_n \in T_{\leftarrow}$ to a constraint $(S \Vdash u) \in C$. If there exists a most general unifier τ of u and t, then the constraint is replaced by n constraints $(S \Vdash t_1, \cdots, S \Vdash t_n)\tau$. The properties of $T_{\leftarrow}$ ensure that rule (3) can be applied only finitely many times (cf. Theorem C.4). The partial substitution τ is also applied to the remaining (if any) constraints in C.

Let $\rightsquigarrow^*$ be the reflexive and transitive closure of $\rightsquigarrow$. We write $C \rightsquigarrow^*_\tau C'$ to denote that there exist constraint systems $C_1, \cdots, C_k$ and substitutions $\tau_1, \cdots, \tau_k$ (for $k \geq 0$) such that $C \rightsquigarrow_{\tau_1} C_1 \cdots \rightsquigarrow_{\tau_k} C_k$, $C_k = C'$ and $\tau = \tau_1 \cdots \tau_k$. If $C_0 \rightsquigarrow^*_\tau \emptyset$ then all leaves of a tentative proof tree have been discharged; hence *ps* returns the solution τ. [6] If $C_0 \rightsquigarrow^*_\tau C'$, with C' being a non-empty constraint system not further reducible, then the tentative proof at hand fails, and *ps* backtracks (i.e. *ps* tries different sequences of applications of the rules of Figure 3). If no sequence of rule applications leads to an empty constraint system, then *ps* reports that the constraint system C_0 has no solutions.

Observe that *ps* is non-deterministic, as there can be multiple solutions for the input constraint $\Omega \Vdash u$. Intuitively, for any possible "proof skeleton" Π of $\Omega \Vdash u$, the procedure *ps* returns a "maximal" solution τ: if τ' is a solution such that $\Pi\tau'$ is a valid proof tree, and $\tau' \sqsubseteq \tau$, then $\tau \equiv \tau'$; that is, τ and τ' are equivalent modulo alpha renaming of the variables. There are finitely many such maximal solutions for any constraint $\Omega \Vdash u$ (namely one for each possible proof skeleton).

In the following sections, we prove that the proof search procedure *ps* is terminating, sound and complete. The proofs exploit the fact that every ground deduction relation $\vdash^T$ induced by a Horn theory T respects Tarski's conditions for consequence relations, namely:

1. *Set membership*: if $u \in S$, then $S \vdash^T u$;
2. *Monotonicity*: if $S \vdash^T u$ and $S \subseteq S'$, then $S' \vdash^T u$;
3. *Consequence*: if $S \vdash^T u$ and $S \cup \{u\} \vdash^T v$, then $S \vdash^T v$.

As the **DC** fragment allows only the unary predicate $\mathcal{K}$ (see § 4) all atoms are of the form $\mathcal{K}(t)$, with t being a term; in order to simplify the presentation, in the following sections we will refer to the argument t rather than to the atom $\mathcal{K}(t)$, i.e. we will reason about terms rather than atoms.

[6] More precisely, if τ is the substitution found such that $C_0 \rightsquigarrow^*_\tau \emptyset$, *ps* returns the substitution $\tau' = \{X \mapsto t \mid (X \mapsto t) \in \tau^*, X \text{ is a variable occurring in the specification}\}$, with τ^* being the least fixpoint of the substitution τ. The reason for this is twofold: from a conceptual point of view, variables introduced by application of the policy rules in the evaluation of a guard are not under the control of the attacker nor of the participant, hence their presence is irrelevant; from a technical point of view, returning a substitution τ that instantiates variables not appearing in the specification would complicate the reasoning and the exposition of our technique to check satisfiability of negative constraints, cf. Appendix C.5.

Termination

Theorem 5. *The proof search procedure ps terminates for all deduction problems in* **AL**.

Proof. To prove termination of the *ps* procedure we define two measures, $\omega_{\leftarrow}$ and $\omega_{\rightarrow}$, for terms in $\mathcal{T}_{\Sigma_{infon}(\mathcal{V})}$, as follows:

$$\omega_{\rightarrow}(u) = \begin{cases} \omega_{\rightarrow}(t\tau) + 1 \text{ if } \exists t \leftarrow t_1, \cdots, t_n, a \in T_{\rightarrow}, \tau = mgu(u, a) \\ 1 \qquad\qquad \text{otherwise} \end{cases}$$

$$\omega_{\leftarrow}(u) = \begin{cases} \max_{R \in T_{\leftarrow}} \omega_R(u) + 1 \text{ if } T_{\leftarrow} \neq \emptyset \\ 0 \qquad\qquad\qquad\quad \text{otherwise} \end{cases}$$

$$\omega_R(u) = \begin{cases} \max_{i \in [1,n]} \omega_{\leftarrow}(t_i\tau) \text{ if } R = t \leftarrow t_1, \cdots, t_n, \tau = mgu(t, u) \\ 0 \qquad\qquad\qquad\quad \text{otherwise} \end{cases}$$

Intuitively, $\omega_{\rightarrow}(u)$ is a measure of how many applications of a rule $R : t \leftarrow t_1, \cdots, t_n, a \in T_{\rightarrow}$ can stem from unifying a term $u \in \mathcal{T}_{\Sigma_{infon}(\mathcal{V}_{msg})}$ (that is, a term of type infon containing only variables of type msg) with the anchor a of rule R; we informally refer to it as the *anchor potential* of u. We extend $\omega_{\rightarrow}$ to sets of terms, so that $\omega_{\rightarrow}(\{u_1, \cdots, u_n\}) = \sum_{i \in [1,n]} \omega_{\rightarrow}(u_i)$.

The function $\omega_{\leftarrow}$ is a measure of the length of the longest reduction sequence induced by u, under theory $T_{\leftarrow}$; we informally refer to it as the *rewriting potential* of u.

The function $\omega_{\rightarrow}$ is well defined due to the conditions on **AL** theories expressed in Section 4: since the rewrite system $\{a \Rightarrow t\}$ (for each $t \leftarrow t_1, \cdots, t_n, a \in T_{\rightarrow}$) is terminating, there is no infinite sequence of rewriting steps. Observe that considering such rewrite system is sufficient as the anchors of other rules in $T_{\rightarrow}$ can not unify with t (cf. § 4). Similarly for function $\omega_{\leftarrow}$, the rewrite system $\mathfrak{R}_{T_{\leftarrow}} = \{t \Rightarrow t_1, \cdots, t_n \mid t \leftarrow t_1, \cdots, t_n \in T_{\leftarrow}\}$ is terminating, hence there is no infinite sequence of rewriting steps.

We now define a weight function ω for constraints. For a constraint $S \Vdash u$, $\omega(S \Vdash u) = (\omega_{\rightarrow}(S), \omega_{\leftarrow}(u))$; that is, the weight of $S \Vdash u$ is the pair whose first element is the anchor potential of the set S and whose second element is the rewriting potential of the term u. We show that at each derivation step, if a constraint C is removed and constraints $C_1, \cdots, C_k$ are introduced, $\omega(C) > \omega(C_i)$ for all $i \in [1, k]$, according to the usual lexicographical order. We condition on the reduction rule applied:

(1) $S \sqcup \{t\} \Vdash u \rightsquigarrow_{\tau} \emptyset$, with $\tau = mgu(u, t)$. In this case the claim is trivially satisfied.
(2) $S \sqcup \{z\} \Vdash u \rightsquigarrow_{\tau} S\tau \Vdash t_1\tau, \cdots, S \Vdash t_n\tau, S\tau \cup \{t\tau\} \Vdash u\tau$, with $\tau = mgu(z, a)$, for some $R : t \leftarrow t_1, \cdots, t_n, a \in T_{\rightarrow}$. For any constraint $S\tau \Vdash t_i\tau$, with $i \in [1, n]$, by Lemma 3 $\omega_{\rightarrow}(S\tau) \leq \omega_{\rightarrow}(S)$, and $\omega_{\rightarrow}(S \sqcup \{z\}) > \omega_{\rightarrow}(S)$ because $\omega_{\rightarrow}(z) > 0$, hence $\omega(S \sqcup \{z\} \Vdash u) > \omega(S\tau \Vdash t_i\tau)$. Similarly for the constraint $S\tau \cup \{t\tau\} \Vdash u\tau$, $\omega_{\rightarrow}(S\tau \cup \{t\tau\}) = \omega_{\rightarrow}(S \cup \{t\tau\})$, and since $\omega_{\rightarrow}(z) = \omega_{\rightarrow}(t\tau) + 1$ by definition of $\omega_{\rightarrow}$ then $\omega(S \sqcup \{z\} \Vdash u) > \omega(S\tau \cup \{t\tau\} \Vdash u\tau)$.

(3) $S \Vdash u \rightsquigarrow_\tau S\tau \Vdash t_1\tau, \cdots, S\tau \Vdash t_n\tau$, with $\tau = mgu(u, t)$ for some $t \leftarrow t_1, \cdots, t_n \in T_\leftarrow$. For any constraint $S\tau \Vdash t_i\tau$, with $i \in [1, n]$, by Lemma 3 $\omega_\rightarrow(S) = \omega_\rightarrow(S\tau)$, and by Lemma 4 $\omega_\leftarrow(u) > \omega_\leftarrow(t_i) \leq \omega_\leftarrow(t_i\tau)$ by definition of $\omega_\leftarrow$, hence $\omega(S \Vdash u) > \omega(S\tau \Vdash t_i\tau)$.

We consider now a set of constraints. We use the following observations to assert that the *ps* procedure terminates on the constraint set. Consider the derivation step $CS \sqcup \{C\} \rightsquigarrow_\tau CS\tau \cup C'$:

- Following from Lemmas 3 and 4, $\omega(CS) \geq \omega(CS\tau)$; that is, when a reduction step is performed on a constraint in the constraint system, and the resulting solution τ applied to the remainder of the constraint system, the weight of the remainder of the constraint system is not increased.
- C' is always a finite set of constraints (obvious because of the finite number of premises of the rules in T), hence the derivation tree stemming from C is finitely branching;
- the derivation tree stemming from C has finite depth, which is implied by the fact that $\omega(C) > \omega(C'')$ for any $C'' \in C'$, shown above.

The considerations above show that all possible derivations of the constraint system CS terminate.

(end of proof)

For the following lemmas, we observe that every application of a rule in T uses fresh variables, that is, they do not appear elsewhere. Consequently, they do not appear in the domain of any substitution σ prior to consideration of that instance of the rule; in particular, if t is a term appearing in a rule, $t\sigma = t$.

Lemma 3. *Let $u \in \mathcal{T}_{\Sigma_{infon}(\mathcal{V}_{msg})}$ be a term of type* infon*, containing only variables of type* msg*, and σ be any well typed substitution. Then $\omega_\rightarrow(u) \geq \omega_\rightarrow(u\tau)$.*

Proof. We prove the claim by induction on the value of $\omega_\rightarrow(u)$. If $\omega_\rightarrow(u) = 0$, then there is no rule $t \leftarrow t_1, \cdots, t_n, a \in T_\rightarrow$ and no substitution τ such that $u\tau = a\tau$. Assume, by contradiction, that $\omega_\rightarrow(u\sigma) > 0$. Then there should exists τ such that $u\sigma\tau = a\tau = a\sigma\tau$, hence $\sigma\tau$ would be a unifier for u and a, contradicting the hypothesis.

If $\omega_\rightarrow(u) > 0$, then $\omega_\rightarrow(u) = \omega_\rightarrow(t\tau) + 1$, for some rule $t \leftarrow t_1, \cdots, t_n, a \in T_\rightarrow$ and $\tau = mgu(u, a)$. We condition on whether $u\sigma$ unifies with a, for some $t \leftarrow t_1, \cdots, t_n, a \in T_\rightarrow$. If there exists no τ' such that $u\sigma\tau' = a\tau'$, then $\omega_\rightarrow(u\sigma) = 0 \leq \omega_\rightarrow(u)$. If there exists τ' such that $u\sigma\tau' = a\tau'$, by the consideration above $u\sigma\tau' = a\sigma\tau'$. Then $\sigma\tau'$ is a unifier for u and a, and therefore there exists σ' such that $\sigma\tau' = \tau\sigma'$. Then $\omega_\rightarrow(u\sigma) = \omega_\rightarrow(t\sigma\tau') + 1 = \omega_\rightarrow(t\tau\sigma') + 1$. By Lemma 5 τ contains all variables of a and maps them to terms containing only variables of type msg, and $var(t) \subseteq var(a)$ (by definition of **AL**, cf. § 4), then $t\tau$ is a term containing only variables of type msg. By induction hypothesis, $\omega_\rightarrow(t\tau\sigma') \leq \omega_\rightarrow(t\tau)$, so $\omega_\rightarrow(u\sigma) = \omega_\rightarrow(t\tau\sigma') + 1 \leq \omega_\rightarrow(t\tau) + 1 = \omega_\rightarrow(u)$.

Lemma 4. *Let $u \in \mathcal{T}_{\Sigma_{infon}(\mathcal{V})}$ be a term of type* infon*, and σ be any well typed substitution. Then $\omega_\leftarrow(u) \geq \omega_\leftarrow(u\tau)$.*

Proof. We prove the claim by induction on the value of $\omega_{\leftarrow}(u)$. If $\omega_{\leftarrow}(u) = 0$, then there is no rule $t \leftarrow t_1, \cdots, t_n \in T_{\leftarrow}$ and no substitution τ such that $u\tau = t\tau$. Assume, by contradiction, that $\omega_{\leftarrow}(u\sigma) > 0$. Then there should exists τ such that $u\sigma\tau = t\tau = t\sigma\tau$, hence $\sigma\tau$ would be a unifier for u and t, contradicting the hypothesis.

Assume now that $\omega_{\leftarrow}(u) > 0$. We condition on whether $u\sigma$ unifies with a, for some $t \leftarrow t_1, \cdots, t_n \in T_{\leftarrow}$. If there exists no τ' such that $u\sigma\tau' = t\tau'$, then $\omega_{\leftarrow}(u\sigma) = 0 \leq \omega_{\leftarrow}(u)$. If there exists τ' such that $u\sigma\tau' = t\tau'$ for some rule $t \leftarrow t_1, \cdots, t_n \in T_{\leftarrow}$; we assume, without loss of generality, that $t \leftarrow t_1, \cdots, t_n \in T_{\leftarrow}$ is the rule with the premise t_i for which $\omega_{\leftarrow}(t_i\sigma\tau')$ is maximal. By the consideration above $u\sigma\tau' = t\tau' = t\sigma\tau'$, hence $\sigma\tau'$ is a unifier for u and a, and there exists $\tau = mgu(u, t)$ and σ' such that $\sigma\tau' = \tau\sigma'$. By definition of $\omega_{\leftarrow}$, $\omega_{\leftarrow}(u) \geq \omega_{\leftarrow}(t_i\tau) + 1$. Then $\omega_{\rightarrow}(u\sigma) = \omega_{\rightarrow}(t_i\sigma\tau') + 1 = \omega_{\rightarrow}(t_i\tau\sigma') + 1$, and by induction hypothesis $\omega_{\rightarrow}(t_i\tau\sigma') \leq \omega_{\rightarrow}(t_i\tau)$, hence $\omega_{\rightarrow}(u\sigma) \leq \omega_{\rightarrow}(t_i\tau) + 1 \leq \omega_{\rightarrow}(u)$.

Lemma 5. *Let $a \in \mathcal{T}_{\Sigma_{infon}(\mathcal{V}_{infon})}$ and $z \in \mathcal{T}_{\Sigma_{infon}(\mathcal{V}_{msg})}$. If there exists $\tau = mgu(a, z) = \{X_1 \mapsto t_1, \cdots, X_n \mapsto t_n\}$ then:*

1. *for all $X \in var(z) \cap \{X_1, \cdots, X_n\}$ $X\tau$ is ground;*
2. *$var(a) \subseteq \{X_1, \cdots, X_n\}$;*
3. *for all $X \in var(a)$ $var(X\tau) \cap \mathcal{V}_{infon} = \emptyset$.*

Proof. We start by proving (1). Let $X \in var(z) \cap \{X_1, \cdots, X_n\}$; then X maps to a subterm t of a of type msg. t can not be a variable, because variables in a are of type infon; also, t can not contain any variables, because terms of type msg can not contain variables of type infon (cf. § 4).

We now prove (2). Assume, by contradiction, that there exists $X \in var(a)$ such that $X \notin \{X_1, \cdots, X_n\}$. But then there must exist $Y \in var(z) \cap \{X_1, \cdots, X_n\}$ such that $X \in var(Y\tau)$ (i.e. X is a subterm of the subterm of a that Y maps to); because of (1), this is not possible because Y must map to a ground term. Therefore, $X \in \{X_1, \cdots, X_n\}$.

Finally, (3) follows trivially from (2) and the assumption that z does not contain any variable of type infon.

Soundness

Theorem 6. *The proof search procedure ps is sound. That is, for two constraint systems C and C' and a substitution ρ that is a solution for C', if $C \rightsquigarrow_\tau C'$ then $\tau\rho$ is a solution for C.*

Proof. We condition on the rule applied:

(1) $C \cup \{S \Vdash u\} \rightsquigarrow_\tau C\tau$, with $\tau = mgu(u, t)$ for some $t \in S$. We need to show that $\tau\rho$ is a solution for $S \Vdash u$. Since $u\tau = t\tau$ and $t\tau \in S\tau$, it follows that $S\tau \vdash u\tau$, and obviously also $S\tau\rho \vdash u\tau\rho$.

(2) $C \cup \{S \cup \{z\} \Vdash u\} \rightsquigarrow_\tau (C \cup \{S \Vdash t_1, \cdots, S \Vdash t_1, S \cup \{t\} \Vdash u\})\tau$, for $\tau = mgu(z, a)$ with a being the anchor of a rule $t \leftarrow t_1, \cdots, t_n, a \in T_\rightarrow$. We need to prove that $S\tau\rho \vdash t_1\tau\rho, \cdots, S\tau\rho \vdash t_n\tau\rho, S\tau\rho \cup \{t\tau\rho\} \vdash u\tau\rho$ implies $S\tau\rho \cup \{z\rho\} \vdash u\rho$. Due to monotonicity of consequence relations (property (2) of Tarski's) $S\tau\rho \cup \{z\tau\rho\} \vdash t_1\tau\rho, \cdots, S\tau\rho \cup \{z\tau\rho\} \vdash t_n\tau\rho, S\tau\rho \cup \{z\tau\rho, t\tau\rho\} \vdash u\tau\rho$. Furthermore, since $z\tau = a\tau$ then $S\tau\rho \cup \{z\tau\rho\} \vdash a\tau\rho$. We can therefore apply rule $t \leftarrow t_1, \cdots, t_n, a$ to conlude that $S\tau\rho \cup \{z\tau\rho\} \vdash t\tau\rho$. Then, by property (3) of Tarski's consequence relations, it follows that $S\tau\rho \cup \{z\tau\rho, t\tau\rho\} \vdash u\rho$ implies $S\tau\rho \cup \{z\tau\rho\} \vdash u\rho$.

(3) $C \cup \{S \Vdash u\} \rightsquigarrow_\tau C\tau \cup \{S\tau \Vdash p_1\tau, \cdots, S\tau \Vdash p_n\tau\}$, with $\tau = mgu(u, p)$ for some $p \leftarrow p_1, \cdots, p_n \in T_\leftarrow$. We need to show that $S\tau\rho \vdash p_1\tau\rho, \cdots, S\tau\rho \vdash p_n\tau\rho$ implies $S\tau\rho \vdash u\tau\rho$. Since $p\tau = u\tau$, this equals to implying $S\tau\rho \vdash p\tau\rho$. By application of rule $p \leftarrow p_1, \cdots, p_n$ the implication is trivially shown.

Completeness

Theorem 7. *The proof search procedure ps is complete. That is, for a (reducible) constraint systems C and a substitution ω that is a solution for C, there exists a constraint system C' such that $C \rightsquigarrow_\tau C'$ and $\tau \sqsubseteq \omega$.*

Proof. Consider a constraint $S \Vdash u \in C$, and a proof Π of $S\omega \vdash u\omega$. The proof goes by induction on the depth of Π. We condition on the last rule applied in Π:

- If the last rule applied is the axiom (i.e. the conclusion $u\omega$ belongs to the set of axioms $S\omega$), then $u\omega = t\omega \in S\omega$. Therefore rule (1) of ps is applicable, and since $\tau = mgu(u, t)$ by definition of mgu there exists a substitution ρ such that $\omega = \tau\rho$.
- If the last rule applied is $t \leftarrow t_1, \cdots, t_n, a \in T_\rightarrow$ (with a being the anchor of the rule), then due to Lemma 6 (see below) rule (2) of ps is applicable for a term $z \in S$ such that $z\omega = a\omega$. In this case the constraint $S \Vdash u$ is replaced by the constraints $S'\tau \Vdash t_1\tau, \cdots, S'\tau \Vdash t_n\tau, S'\tau \cup \{t\tau\} \Vdash u\tau$, with $S' = S \setminus \{z\}$. Since $\tau = mgu(a, z)$ a solution ω refines τ. Again due to Lemma 6 $S\omega \vdash t_1\omega, \cdots, S\omega \vdash t_n\omega$ and $S\omega \cup \{t\omega\} \vdash a\omega$.
- If the last rule applied is a rule $p \leftarrow p_1, \cdots, p_n \in T_\leftarrow$, then $S\omega \vdash p_1\omega, \cdots, S\omega \vdash p_n\omega$ and $u\omega = p\omega$. Rule (3) is then applicable. Since $\tau = mgu(u, p)$, by definition of mgu there exists ρ such that $\omega = \tau\rho$. Since $S\omega \vdash p_1\tau\rho, \cdots, S\omega \vdash p_n\tau\rho$ then ρ is a solution for the constraints $S\tau \Vdash p_1\tau, \cdots, S\tau \Vdash p_n\tau$.

For the following lemmas, we remark that a normal proof Π is a proof where a label $A \vdash c$ never appears more than once on the same branch of Π. (cf. § 4), an

Lemma 6. *Let T be an* **AL** *theory, $\vdash$ the ground deduction relation induced by T, S a set of terms, u a term, and ω a substitution such that $S\omega \vdash u\omega$. If there exists a normal proof of $S\omega \vdash u\omega$ ending with an instance $t' \leftarrow t'_1, \cdots, t'_n, a'$ of rule R of rule $R : t \leftarrow t_1, \cdots, t_n, a \in T_\rightarrow$, then there exists an instance $t'' \leftarrow t''_1, \cdots, t''_n, a''$ of rule R and a term $z \in S$ such that $z\omega = a''\omega$, $S'\omega \vdash t''_1\omega, \cdots, S'\omega \vdash t''_n\omega$ and $S'\omega \cup \{t''\omega\} \vdash u\omega$, with $S' = S \setminus \{z\}$.*

Proof. Let Π be a normal proof of $S\omega \vdash u\omega$ that ends with an instance $t' \leftarrow t'_1, \cdots, t'_n, a'$ of rule R. Then $u\omega = t'\omega$ and $S\omega \vdash t'_1\omega, \cdots, S\omega \vdash t'_n\omega, S\omega \vdash a'\omega$, which by property (3) of Tarski's consequence relations implies $S\omega \vdash t'\omega = u\omega$.

Let there be $z \in S$ such that $z\omega = a'\omega$. Then, by Lemma 7, $S'\omega \vdash t'_1\omega, \cdots, S'\omega \vdash t'_n\omega$, with $S' = S \setminus \{z\}$. Also, $S\omega = (S' \cup \{a'\})\omega \vdash t'\omega = u\omega$. Hence the statement of the lemma holds for $a' = a'', t' = t'', t'_1 = t''_1, \cdots, t'_n = t''_n$.

Assume now there is no $z \in S$ such that $z\omega = a'\omega$, and Π' be a proof of $S\omega \vdash a'\omega$. It is immediate to see that Π' can not end with the axiom rule. Furthermore, by definition of **AL** (cf. § 2) a' can only unify with the head of rule R. Then by induction hypothesis there exists an instance $t'' \leftarrow t''_1, \cdots, t''_n, a''$ of R and a term $z \in S$ such that $z\omega = a''\omega$, $S'\omega \vdash t''_1\omega, \cdots, S'\omega \vdash t''_n\omega$ and $S'\omega \cup \{t''\omega\} \vdash a'\omega$, with $S' = S \setminus \{z\}$.

To conclude the proof it remains to show that $S'\omega \vdash t'_1\omega, \cdots, S'\omega \vdash t'_n\omega$. Consider t'_i among $t'_1, \cdots, t'_n$. Assume by contradiction that $S\omega \vdash t'_i\omega$ but $(S \setminus \{z\})\omega \nvdash t'_i\omega$. Then the subproof Π' of $S\omega \vdash t'_i\omega$ has in its leaves (at least) an application of the axiom rule unifying z with a premise p of a rule R' (in the context of proof Π). By definition of **AL**, p can not be a premise of any rule in $T \setminus \{R\}$, nor any of the premises $t_1, \cdots, t_n$ in rule R. Then p is the anchor of an instance of rule R. Observe that the head t of rule R unifies with anchor a (otherwise there would be $z \in S$ such that $z\omega = a'\omega$). Consequently, each application of rule R yields a term that unifies with the anchor a, which can therefore only be used as anchor for another application of R. This implies that there is path in Π from z to t'_i consisting of (zero or more) applications of rule R, the last of which yields an atom that unifies with t'_i. But this leads to a contradiction with respect to the assumed property that anchor a does not unify with any of the premises $t_1, \cdots, t_n$. It must then be the case that $S'\omega \vdash t'_i\omega$. would of This

Finally, by monotonicity of consequence relations (property 2 of Tarski's) we have $S'\omega \cup \{t'\omega\} \vdash t_1\omega, \cdots, S'\omega \cup \{t'\omega\} \vdash t_n\omega$. Then by applying R we obtain that $S'\omega \cup \{t'\omega\} \vdash t\omega = u\omega$, thus completing the proof.

Lemma 7. *Let T be an* **AL** *theory, $\vdash$ the ground deduction relation induced by T, S a set of terms, u a term, and ω a substitution such that $S\omega \vdash u\omega$. If there exists a normal proof of $S\omega \vdash u\omega$ whose last rule is $t \leftarrow t_1, \cdots, t_n, a \in T_{\rightarrow}$ and there exists $z \in S$ such that $z\omega = a\omega$, then $S'\omega \vdash t_1\omega, \cdots, S'\omega \vdash t_n\omega$, with $S' = S \setminus \{z\}$.*

Proof. Let Π be a normal proof of $S\omega \vdash u\omega$ ending with the application of rule $R : t \leftarrow t_1, \cdots, t_n, a \in T_{\rightarrow}$. Let also $z \in S$ such that $z\omega = a\omega$. Assume by contradiction that there is t_i among premises $t_1, \cdots, t_n$ such that $(S \setminus \{z\})\omega \nvdash t_i\omega$. Then for any proof Π' of $S\omega \vdash t_i\omega$ there is a leaf of Π labeled with an application of the axiom rule that equals z and some premise p of a rule R in T, under substitution ω. If p is a premise of a rule in $T \setminus \{R\}$ or one of the premises $t_1, \cdots, t_n$ of rule R, then $p\omega = z\omega = a\omega$, which contradicts the hypothesis that a does not unify with any other premise in T (cf. def. 2). If $p = a'$ for an instance $t' \leftarrow t'_1, \cdots, t'_n, a'$ of rule R, since $var(t') \subseteq var(a')$ then $t'\omega = t\omega$, thus contradicting the normality assumption of proof Π.

C.5 Solving negative constraints

We explain here the procedure *solveALNegative* to solve the set $N = neg(cs(\iota, G))$ of negative constraints in $cs(\iota, G)$. Without loss of generality, we assume that the procedure *solveALNegative* is invoked after that $att(cs(\iota, G))$ has been reduced to a trivial form, and $pos(cs(\iota, G))$ has been reduced to the empty set (*solve* always allows such a reduction path). If there exists a solution ρ of N, then *solveALNegative* returns T and outputs ρ. The procedure returns F if there exists no solution for $neg(cs(\iota, G))$. In the following we give an informal description of the procedure.

Let $c = T \Vdash u$ be a positive constraint. We define the *complement* $\neg c$ of c as the negative constraint $T \not\Vdash u$. Symmetrically, the complement $\neg c$ of a negative constraint $c = T \not\Vdash u$ is the positive constraint $T \Vdash u$. Clearly $\neg\neg c = c$ for any constraint c. Intuitively, a constraint c is satisfied if its complement $\neg c$ is not satisfied; therefore, a constraint is valid iff its complement is not satisfiable. We extend the definition of complement to constraint systems: for a constraint system $C = \{c_1, \cdots, c_n\}$, the complement of C is the constraint system $\neg C = \{\neg c_1, \cdots, \neg c_n\}$.

Consider the negative constraint $c = T \not\Vdash u$. The procedure *solveALNegative* tries to construct a substitution ρ such that $\neg c\rho$ fails, i.e. $\neg c\rho$ has no solutions. Indeed, $\neg c\rho$ fails if $\not\exists \rho'.\ T\rho\rho' \vdash u\rho\rho'$, which makes ρ a solution for c. The solution ρ should differ from each substitution τ returned by *ps* for $\neg c$; that is, $X\rho \neq X\tau$ for at least one variable X in the intersection of the domains of ρ and τ. This procedure is described in pseudocode in Figure 3.

Algorithm 3 Procedure *solveALNegative*

```
REQUIRES: N = {c_1, ··· , c_k}
  δ:=∅
  for all c ∈ N do
    repeat
      τ := ps(¬c)
      if τ = ∅ then
        return (F,∅)
      else
        δ:=δ ∪ τ
    until ps(¬c) has no more solutions
  φ = X_1 ≠ t_1 ∧ ··· ∧ X_n ≠ t_n, for δ = {(X_1, t_1), ··· , (X_n, t_n)}
  ρ:=GDPLL(φ)
  return (T,ρ)
```

We employ a binary relation $\delta : \mathcal{V} \to \mathcal{T}_{\Sigma(\mathcal{V})}$ to store all solutions for the constraints in $\neg N$. These substitutions will be later used to provide a witness. For each constraint $c \in N$, *solveALNegative* repeatedly invokes the **AL** proof search procedure *ps* on $\neg c$, until no more solutions for $\neg c$ are available. If a substitution τ returned by *ps* is the empty substitution, then *solveALNegative*

returns F (see Theorem 8). If τ is not the empty substitution, then τ is added to δ. If no solution for the constraints in $\neg N$ is the empty substitution, then there exists a solution ρ of N (cf. Theorem 8). To provide the witness ρ we can represent the relation δ as a formula ϕ in equality logic; in particular, as a conjunction of disequalities over terms. Then existing algorithms (e.g. the GDPLL algorithm of [5]) can be used to find an instantiation of the variables that satisfies ϕ.

Termination of the procedure *solveALNegative* follows trivially from the termination of *ps* (cf. Appendix C.4), the finite number of results yielded by *ps*, and the finiteness of the set of negative constraints in input. The soundness and completeness of the procedure are implied by the following theorem.

Theorem 8. *Let N be a finite set of negative policy constraints and $\tau_1, \cdots, \tau_\ell$ be the substitutions returned by ps for the constraints in $\neg N$. Then there exists a solution ρ for N if and only if $\tau_i \neq \emptyset$ for all $i \in [i, \ell]$.*

Proof. Let $N = \{T_1 \not\Vdash u_1, \cdots, T_k \not\Vdash u_k\}$ and $\tau_1, \cdots, \tau_\ell$ be all the substitutions returned by *ps* for the constraints in $\neg N$; that is, for all $i \in [1, \ell]$, there exists $j \in [1, k]$ such that $T_j\tau \vdash u_j\tau$ for any total grounding substitution $\tau \sqsupseteq \tau_i$. The proof is split in two directions:

- $\Rightarrow$ Let us represent the solutions $\tau_1, \cdots, \tau_\ell$ as conjunctions of equations between terms, so that each τ_i is of the form $X^i_1 = t^i_1 \wedge \cdots \wedge X^i_{n^i} = t^i_{n^i}$ for some $X^i_1, \cdots, X^i_{n^i} \in \mathcal{V}$ and some $t^i_1, \cdots, t^i_{n^i} \in \mathcal{T}_{\Sigma(\mathcal{V})}$. A solution for N is a mapping ρ that differs from each of the solutions for $\neg N$ for at least one variable assignment. In particular, if ρ differs from $\tau_1, \cdots, \tau_\ell$ for all variable assignments ρ is a solution for N. Thus, it suffices to show that there is an assignment which is a model of the formula $\phi = \neg(X^1_1 = t^1_1) \wedge \cdots \wedge \neg(X^\ell_{n^\ell} = t^\ell_{n^\ell})$. Assume such an assignment does not exist. Then the formula $\neg\phi = X^1_1 = t^1_1 \vee \cdots \vee X^\ell_{n^\ell} = t^\ell_{n^\ell}$ is valid, i.e., any assignment is a model. Since $\mathcal{T}_{\Sigma(\emptyset)}$ is infinite, this would be possible only if $\neg\phi$ had an infinite number of disjuncts. This proves that there must exist a model ρ for ϕ, which is also a solution for N.
- $\Leftarrow$ Assume, by contradiction, that there exists a solution ρ of N and $\tau_i = \emptyset$, for some $i \in [1, \ell]$. Since ρ solves N, then $T_1\rho \not\vdash u_1\rho, \cdots, T_k\rho \not\vdash u_k\rho$. If $\tau_i = \emptyset$, then $T_j\tau \vdash u_j\tau$, for some $j \in [1, k]$ and any total grounding substitution τ. In particular, $T_j\rho \vdash u_j\rho$, which contradicts the hypotheses.

C.6 Symbolic treatment of retraction

We present now a technique to handle retracted facts symbolically, that integrates with the procedure Reach (Algorithm 1) and its subroutines. We start with an example to show some of the subtleties entailed by the retraction of facts. Consider a process π, with an empty policy, empty initial extensional

knowledge Ω, and a thread as follows:

$$\begin{array}{ll} (1) & \mathsf{rcv}(X) \blacktriangleright (\{f(X)\}, \emptyset) \\ (2) & \mathsf{rcv}(Y) \blacktriangleright (\{g(Y)\}, \{f(Y)\}) \\ (3) & (\{f(Z)\}, \emptyset) \blacktriangleright \mathsf{snd}(t_1) \\ (4) & (\{g(Z)\}, \emptyset) \blacktriangleright \mathsf{snd}(t_2) \end{array}$$

Evaluation of the guard at line (3) is conditional to the concrete values of the variables X and Y: if $Y = X$ then π's extensional knowledge at line (3) is $\Omega = \{f(X)\} \setminus \{f(X)\} = \emptyset$, and therefore $\Omega \nvdash f(Z)$; on the other hand, if $Y \neq X$ then π's extensional knowledge at line (3) is $\Omega = \{f(X)\} \setminus \{f(Y)\} = \{f(X)\}$, from which $f(Z)$ can be inferred for $X = Z$. The guard at line (4) evaluates to true only if $Y = Z$, but such variables instantiation can not be allowed, as it would entail $Y = Z = X$, which is required not to happen for the guard at line (3) to evaluate to true. Consequently, the above thread is not executable, because the guards at lines (3) and (4) can never evaluate both to true.

Let cap be a CAP in $\mathbf{DC}$, $\iota = z_0 e_1 z_1 \cdots e_n z_n$ be an interleaving of cap and G be a goal. Upon construction of the constraint system $cs(\iota, G)$ (see Appendix C), we associate to each constraint generated a relation $\Delta : \mathcal{A}_{\Sigma(\mathcal{V})} \times 2^{\mathcal{A}_{\Sigma(\mathcal{V})}}$ between each previously retracted atom r and the set P of atoms in the extensional knowledge (at the moment of retraction) that unify with r; that is, an atom $p \in P$ could be retracted from the extensional knowledge under a substitution σ such that $p\sigma = r\sigma$.

Then procedure *ps* (see Appendix C.4) is modified as follows: it takes as input a constraint $c = \Omega \vdash u$ and the relation Δ associated with c, and returns a substitution σ and a set of disequalities Ψ; when rule 1 is applied, that is, an atom a in Ω is unified with one of the leaves of the tentative proof tree guessed by *ps*, the set of disequalities $\{a \neq r \mid (r, P) \in \Delta, a \in P\}$ is added to Ψ. Intuitively, the disequalities in Ψ at the end of the execution of *ps* are the negation of the equalities that would entail the retraction of some leaves in the guessed proof, and would thus invalidate it.

Also procedure *solveALNegative* (see Appendix C.5) must then be modified accordingly. An immediate observation is that a negated constraint $\neg c$ is unsatisfiable if $ps(c)$ (the call to *ps* with the complement of $\neg c$ in input) returns (σ, Ψ) with σ being the empty substitution and $\Psi = \emptyset$. The intuition behind this observation is similar to what expressed in Appendix C.5: if there exists a proof for the complement c of the negative constraint $\neg c$ that requires no instantiation of cap's variables and can not be invalidated by retraction of some fact, then all substitutions are "valid" refinements of this solution, meaning that there exists no solution for the constraint $\neg c$. Unfortunately, the opposite implication does not hold: it does not suffice that all solutions (σ, Ψ) returned by *ps* for c be such that at least one of σ and Ψ is non-empty, to conclude that $\neg c$ is satisfiable. On the other hand, procedure *solveALNegative* can be modified to solve (when possible) the negative constraints. Let $\{(\sigma_1, \Psi_1), \cdots, (\sigma_k, \Psi_k)\}$ be the set of solutions returned by *ps* for the complement constraint c. Each solution (σ_i, Ψ_i), with $\sigma_i = \{X_1 \mapsto t_1, \cdots, X_n \mapsto t_n\}$ and $\Psi_i = \{a_{11} \neq a_{12}, \cdots, a_{m1} \neq a_{m2}\}$ can

be rewritten into an equivalent (equality logic) formula $\epsilon_i = X_1 = t_1 \wedge \cdots \wedge X_n = t_n \wedge a_{11} \neq a_{12} \wedge \cdots \wedge a_{m1} \neq a_{m2}$. The solutions to the constraint c equal the models of the formula $\phi = \epsilon_1 \vee \cdots \vee \epsilon_k$, thus the solutions for the negated constraint $\neg c$ are the models of the negated formula $\neg\phi = \neg\epsilon_1 \wedge \cdots \wedge \neg\epsilon_k$. The problem of finding the models of $\neg\phi$ is decidable, and can be solved by any existing equality logic solver (e.g. the GDPLL algorithm of [5]).

Finally, for an interleaving ι of cap, *solve* (see Appendix C) stores the union of the disequalities' sets returned by each successful call to *ps*. If *solve* finds a solution σ to the constraint system in input and σ respects the set of disequalities collected *Dis*, then *solve* returns $(\mathsf{T}, \sigma, Dis)$; otherwise, it backtracks in search of other possible solutions. If no substitution σ is found that solves the constraint system and respects the disequalities, then *solve* returns $(\mathsf{F}, \emptyset, \emptyset)$.

C.7 Correctness of the decision procedure for REACH for caps in DC

Finally, in the following sections we prove Theorem 1, by showing that Algorithm 1 is correct; that is, it is sound, complete and terminates in finite time for any CAP in **DC**.

Termination

Theorem 9. *For any input* $\textsc{Reach}\langle \mathsf{cap}, G\rangle$, *with* $\mathsf{cap} = ((\Sigma, \mathcal{F}, \mathcal{V}), \pi_1, \cdots, \pi_\ell, \mathbb{A})$ *being a* CAP *in* **DC**, *and* $G = \pi : a$ *being a goal such that either* $\pi = \mathbb{A}$ *and* $a \in \mathcal{T}_{\Sigma(\emptyset)}$ *or* $\pi \in \{\pi_1, \cdots, \pi_\ell, \mathbb{A}\}$ *and* $a \in \mathcal{A}_{\Sigma(\emptyset)}$, *Algorithm 1 terminates in finite time.*

Proof. It follows trivially from the number of finite interleavings stemming from CAPs in **DC**, and termination of procedure *solve*. Termination of procedure *solve* is also clear from termination of procedures *reduceDY*, *solveALPositive* and *solveALNegative* (cf. Appendices C.3, C.4 and C.5).

Soundness

Theorem 10. *Let* $\mathsf{cap} = ((\Sigma, \mathcal{F}, \mathcal{V}), \pi_1, \cdots, \pi_\ell, \mathbb{A})$ *be a* CAP *in* **DC**, *and* $G = \pi : a$ *be a goal such that either* $\pi = \mathbb{A}$ *and* $a \in \mathcal{T}_{\Sigma(\emptyset)}$ *or* $\pi \in \{\pi_1, \cdots, \pi_\ell, \mathbb{A}\}$ *and* $a \in \mathcal{A}_{\Sigma(\emptyset)}$. *If Algorithm 1 terminates successfully for the reachability problem* $\textsc{Reach}\langle \mathsf{cap}, G\rangle$ *then* $\textsc{Reach}\langle \mathsf{cap}, G\rangle = \mathsf{T}$.

Proof. Let (T, σ) be the return value of Algorithm 1 for the problem $\textsc{Reach}\langle \mathsf{cap}, G\rangle$, for an interleaving of cap $\iota = z_0 e_1 z_1 \cdots e_n z_n$; that is, σ is a solution for $cs(\iota, G)$. Below, we show that, for any ground substitution $\sigma' \sqsupseteq \sigma$, $\iota\sigma'$ is a trace of cap. Recall that we assume that for each guarded send $(g_\exists, g_{\not\exists}) \blacktriangleright \mathsf{snd}(t)$ appearing in the specification of cap only one of $g_\exists$ and $g_{\not\exists}$ is non-empty, and it is a singleton (cf. Appendix C.2).

By definition of interleaving and generation of the corresponding constraint system (cf. Appendix C.2) it is clear that for any solution σ' of the constraint

system $cs(\iota, G)$ generated from interleaving ι, $\iota\sigma''$ is a trace of cap, for a grounding substitution σ'' that refines σ'. The constraint system $cs(\iota, G)$ is partitioned into the sets $A = att(cs(\iota, G))$, $P = pos(cs(\iota, G))$ and $N = neg(cs(\iota, G))$ of attacker constraints, positive policy constraints and negative policy constraints in $cs(\iota, G)$. By soundness of the procedure *reduceDY* (see Appendix C.3), if *reduceDY*(A) terminates successfully returning a substitution σ_1, then any ground refinement of σ_1 is a solution for A. Then by soundness of the procedure *solveALPositive* (see Appendix C.4), if *solveALPositive*$(P\sigma_1)$ terminates successfully returning a substitution σ_2, then any ground refinement of σ_2 is a solution for $P\sigma_2$, and so clearly for $A\sigma_1$. Subsequently, by soundness of the procedure *solveALNegative* (see Appendix C.5), if *solveALNegative*$(N\sigma_1\sigma_2)$ terminates successfully and returns a substitution σ_3, then any ground refinement of σ_3 is a solution for $N\sigma_1\sigma_2$. It follows trivially that the substitution $\sigma = \sigma_1\sigma_2\sigma_3$ returned by algorithm 1 is such that any ground $\sigma' \sqsupseteq \sigma$ is a solution for $cs(\iota, G)$; consequently $\iota\sigma'$ as described above is an interleaving whose ground refinements are traces of cap that satisfy goal G.

Completeness

Theorem 11. *Let* $\mathsf{cap} = ((\Sigma, \mathcal{F}, \mathcal{V}), \pi_1, \cdots, \pi_\ell, \mathbb{A})$ *be a* CAP *in* **DC**, $G = \pi : a$ *being a goal such that either* $\pi = \mathbb{A}$ *and* $a \in \mathcal{T}_{\Sigma(\emptyset)}$ *or* $\pi \in \{\pi_1, \cdots, \pi_\ell, \mathbb{A}\}$ *and* $a \in \mathcal{A}_{\Sigma(\emptyset)}$. *If* $\textsc{Reach}\langle \mathsf{cap}, G\rangle = \mathsf{T}$ *then Algorithm 1 returns* T *along with a witness* $\iota\sigma$. *Here,* ι *is an interleaving of* cap *and* σ *is a solution such that* $\iota\sigma'$ *is a trace of* cap *for any ground substitution* $\sigma' \sqsupseteq \sigma$.

In the following we assume $G = \pi : a$, with $\pi \in \{\pi_1, \cdots, \pi_\ell\}$ and $a \in \mathcal{A}_{\Sigma(\emptyset)}$; the case where $G = \mathbb{A} : m$ with $m \in \mathcal{T}_{\Sigma(\emptyset)}$ is similar. If $\textsc{Reach}\langle \mathsf{cap}, G\rangle = \mathsf{T}$ then there exists in the semantics of cap a trace $\mathfrak{tr} = z_0 e_1 z_1 \cdots e_n z_n$ such that $\Omega_\pi \vdash^\pi a$, with Ω_π being the extensional knowledge of process π in the configuration z_n. By generation of the interleavings, all the interleavings corresponding to a given trace are evaluated by Algorithm 1. For simplicity we assume that for all guarded sends $(g_\exists, g_\nexists) \blacktriangleright \mathsf{snd}(t)$ in cap only one between $g_\exists$ and $g_\nexists$ is non-empty, and it is a singleton; cf. Appendix C.2. It is immediate then that there is a unique interleaving ι and a unique ground substitution σ' such that $\mathfrak{tr} = \iota\sigma'$. The interleaving ι and the goal G are input to the procedure *cs* (see Appendix C.2) which generates a constraint system $cs(\iota)$. Let $A = att(cs(\iota, G))$, $P = pos(cs(\iota, G))$ and $N = neg(cs(\iota, G))$ of attacker constraints, positive policy constraints and negative policy constraints in $cs(\iota, G)$. By generation of the constraints it is clear that σ' is a solution for $cs(\iota, G)$.

The set A is input to procedure *reduceDY*, which by completeness (see Appendix C.3) terminates successfully and returns a partial substitution $\sigma_1 \sqsubseteq \sigma'$. Then the set $P\sigma_1$ is input to *solveALPositive*, which by completeness (see Appendix C.4) terminates successfully and returns a partial substitution $\sigma_1\sigma_2 \sqsubseteq \sigma'$. Finally the set $N\sigma_1\sigma_2$ is input to *solveALNegative*, which by completeness (see Appendix C.5) terminates successfully and returns a partial substitution $\sigma_3 \sqsubseteq \sigma'$. Therefore also algorithm 1 terminates successfully (i.e. returns T) and it is trivial that $\sigma \sqsubseteq \sigma'$, for $\sigma = \sigma_1\sigma_2\sigma_3$.