Evidential Authorization*

Andreas Blass,[1] Yuri Gurevich,[2] Michał Moskal[2] and Itay Neeman[3]

[1] Mathematics, University of Michigan, Ann Arbor, MI, USA
[2] Microsoft Research, Redmond, WA, USA
[3] Mathematics, UCLA, Los Angeles, CA, USA

To Bertrand Meyer, the Eiffel tower of program correctness.

Most fascinating is a feature that would make any journalist tremble. Tuyuca requires verb-endings on statements to show how the speaker knows something. Diga ape-wi means that the boy played soccer (I know because I saw him), while diga ape-hiyi means the boy played soccer (I assume). English can provide such information, but for Tuyuca that is an obligatory ending on the verb. Evidential languages force speakers to think hard about how they learned what they say they know.

—The Economist, January 1, 2010

Abstract. Consider interaction of principals where each principal has its own policy and different principals may not trust each other. In one scenario the principals could be pharmaceutical companies, hospitals, biomedical labs and health related government institutions. In another scenario principals could be navy fleets of different and not necessarily friendly nations. In spite of the complexity of interaction, one may want to ensure that certain properties remain invariant. For example, in the navy scenario, each fleet should have enough information from other fleets to avoid unfortunate incidents. Furthermore, one want to use automated provers to prove invariance. A natural approach to this and many other important problems is to provide a high-level logic-based language for the principals to communicate. We do just that. Three years ago two of us presented the first incarnation of Distributed Knowledge Authorization Language (DKAL). Here we present a new and much different incarnation of DKAL that we call Evidential DKAL. Statements communicated in Evidential DKAL are supposed to be accompanied with sufficient justifications. In general, we construe the term "authorization" in the acronym "DKAL" rather liberally; DKAL is essentially a general policy language. There is a wide spectrum of potential applications of DKAL. One ambitious goes is to provide a framework for establishing and maintaining invariants.

Keywords: access control, authorization, distributed knowledge, evidential, justification, logic-level communication

* Blass was partially supported by NSF grant DMS-0653696. Part of the work reported here was performed during visits of Blass and Neeman to Microsoft.

1 Introduction

Logic-based authorization made its debut in the early 1990's with the Speaks-For calculus [17,1]. The first paper on Distributed-Knowledge Authorization Language (DKAL) was published in 2008 [12]. In the interim, substantial progress had been achieved in logic-based authorization. For brevity, let us just refer to the article [5] on SecPAL (Security Policy Authorization Language) and an informative survey of related work there.

DKAL started as an attempt to extend SecPAL but then the two languages diverged. The main motivation of DKAL was reflected in its title: distributed knowledge. The challenge was how to reason about communicating principals, each computing his own knowledge. The most conspicuous innovation of DKAL vis-à-vis SecPAL was the introduction of logic-level targeted communication among the principals involved in the given scenario. Each principal computes his[4] own knowledge, acts on it, and communicates directly with other principals. No intermediaries are necessary. Autonomous communicating principals had been considered in the literature, in particular in [3] and [8]. Yet, as far as we know, DKAL is the only language in the literature with specific logic-level rules for sending and accepting communications. Many logic-based authorization languages presume the existence of a central engine where all the reasoning occurs. For example, in the world of SecPAL, a central engine controls the resources and serves as the only knowledge manager. The only communications in the language of SecPAL are assertions of the form (P `says` φ), meaning that P says φ to the central engine. In the SecPAL implementation, different central engines do exchange information but at a lower level that is not covered by the language.

Is it important to exchange information among knowledge managers at the logic level? The answer to this question depends on what your goals are. Logic-level communication facilitates reasoning about decentralized systems (see a little example in [15]) and enables automation of such reasoning. It allows you to address new challenges. Imagine a collective of communicating principals in a paranoid world where trust is in short supply. Each principal computes his own knowledge, and not only his data but also his policy is subject to change. And yet some invariant needs to be maintained. For example, there might be just two principals. One of them is a government agency that fiddles with its policy, and the other is a company with continually evolving procedures. The invariant is that the company complies with the agency's requirements. Another example is an e-health ecosystem, and among the invariants are preservation of the safety and the privacy of the patients. Note that logic-level communication may be used to alter authorization policies in a controlled way; a step in that direction was done in [14]. In particular, this enables policy bootstrapping: One principal

[4] Of course, principals do not have to be people. They may be computers, governments, government agencies, companies, universities, etc. But it is convenient to use anthropomorphic terminology.

creates another principal and supplies the newborn with some initial policy, and then the newborn pulls necessary policy extensions from various quarters.

One interesting outcome of logic-level communication is the reemergence of quotations. Quotations were used in Speaks-For but not in recent pre-DKAL logic-based authorization languages. In particular, they are absent in SecPAL. There, communications go to the central engine and are cryptographically signed. Thus, nested quotations are superfluous. What is the point of my telling the central engine that you said φ? It knows. You said φ to it. But in the world where everybody computes his own knowledge, nested quotations are ubiquitous. For example, imagine a committee that conducted a secret vote with different members of the committee residing in different locations. The secretary confirms that two members of the committee said Yes. In a similar vain, targeted communication between principals makes little sense in the world of SecPAL.

In truly decentralized situations, a principal who wants his communications to be believed may need to provide evidence along with the communications. Such evidence may consist merely of signed statements from others but may also involve deductions. This is especially important when the recipient, say a smart card or computer server, needs to be virtually stateless (as far as the interaction in question is concerned) and therefore unable to do much fact checking but can verify justifications that are sent to it. In this paper, we present a version of DKAL, Evidential DKAL, that is geared toward applications where justifications are necessary[5].

We hasten to add that not all applications are of this sort. Imagine that you work for the 911 emergency service. You would not demand that accident reports be cryptographically signed. But you might have rules for assessing the credibility of such reports, even anonymous ones. In general, there are various kinds of applications and ways to assess evidence. In some cases you may rely on a rating system of sorts; in other cases you may decide on the basis of preponderance of evidence. Even gossip may sometimes be informative. For simplicity we treat in this paper only the case where all communications come with justifications; we postpone the general case to future work.

Another innovation of DKAL concerned authorization logic. The Speaks-For calculus is a logic of its own. Later logic-based authorization languages built on Datalog. Datalog achieves its computational efficiency by forbidding the use of functions (it is a purely relational logic apart from constraints formulated in some efficiently decidable theory) and by imposing safety requirements. In essence, the efficiency of Datalog (with or without constraints) results from its small world, consisting of just the constants that appear explicitly. In [12], the underlying logic is an extension of Datalog in which functions are allowed. So the world is no longer small, but restrictions were imposed on the ranges of some variables (so-called regular variables), and then difficult proofs showed that the efficiency is not lost. Subsequently, [14] incorporated the discovery of a strong connection with traditional constructive logic and modified the formulation of DKAL to make the connection conspicuous. In addition it showed that a useful

[5] For evidentiality in natural languages, see [2].

fragment of DKAL logic is decidable in linear time. This modification eventually allowed us to separate concerns and consider logic separately from the rest of the system. Here we treat the underlying logic and additional application-specific assumptions as a parameter.

The present paper is organized as follows. Section 2 introduces an example that will be used throughout the paper. Section 3 is an informal presentation of some of the central ideas of DKAL, which will be developed formally in subsequent sections. The world of DKAL consists of communicating agents. In accordance with the access control tradition, the agents are called principals. The principals perform various computations. Each principal has its own computation state that, in general, evolves in time[6]. Conceptually, the state splits into two parts: substrate and infostrate. Section 4 is devoted to the substrate, and Section 6 is devoted to the infostrate. Between these two, Section 5 is about the logic of DKAL. Section 7 formalizes the policies for the example of Section 2. The final Section 8 is about future work.

Acknowledgment

We are indebted to Roy DSouza, a partner architect at Microsoft, for numerous helpful conversations and for pioneering the use of DKAL in cloud federation scenarios, in particular for compliance. Roy's projects gave the impetus for the development of evidential DKAL.

2 Clinical Trials Scenario, Informal Description

As a running example in this paper, we shall use a partial description of a clinical trial of a new drug. In Section 7, the relevant policies will be written out systematically. In the present section, we describe the background situation and the policies of the parties under consideration.

2.1 Background

A pharmaceutical company developed a certain drug and tested it internally. Before bringing the drug to market, however, it is necessary to conduct a clinical trial, an external study of the safety and efficacy of the drug. Clinical trials are overseen by government regulatory bodies, such as the U.S. Food and Drug Administration (FDA), which eventually decide whether the drug can be marketed.

Although the pharmaceutical company pays for the clinical trial, it does not conduct the trial itself — partly to avoid an appearance of impropriety and partly for the sake of efficiency. The company hires an accredited *contract research organization* to organize and manage the clinical trial. This trial organizer hires

[6] Our notion of state is influenced by the theory of abstract state machines [11] but we do not presume that the reader is familiar with abstract state machines.

sites, like hospitals or clinical labs, to take part in the clinical trial. A site finds appropriate patients and designates physicians, bioscientists, etc., to work on the clinical trial. We presume that the clinical trial employs cloud computing. Patient records and other data relevant to the trial are kept in the cloud, and access to them is controlled by a special key manager.

The conduct of a clinical trial involves the interaction of a great many entities, known as principals in the context of access control. These include government agencies (e.g. FDA), the pharmaceutical company, the trial organizer, the sites, the physicians, the bioscientists, the auditors, and others [10]. A complete description of the policies of all these principals is a major project which we do not undertake here; rather, we use a small part of such a description as an illustration of how policies are handled in DKAL.

2.2 Policies

The policies we use for our example are the ones involved when a physician wants to read the record of a particular patient. These records are stored in the cloud, indexed by patient ID numbers, and, as indicated above, access to patient records is controlled by a key manager. This key manager's role is to check that an access request is legitimate and, if it is, to reveal the necessary cryptographic key to the requester. The legitimacy of a physician's request for the record of a particular patient depends on several factors. We deal with the simple situation where the physician is requesting access to the record of one of his own patients in the trial. Some information in a patient's record should remain inaccessible even to his trial physician, for example, whether the patient is receiving the drug being tested or a placebo. For brevity, we use "record" to mean the part of a patient's record that should be accessible to his trial physician. (Real policies would also have to cover the case where, in an emergency, a physician — any physician — attending the patient may need the full record, including the information that is ordinarily inaccessible.) The key manager does not simply accept the physician's word that this is his patient but requires certification from the site (in this case a hospital or clinic) that assigned that patient (or, more precisely, that patient's ID) to this particular physician. But the key manager may know only the organizers of trials, not the sites they hire. The key manager therefore requires additional certification, from the trial organizer, that the site in question is a legitimate site, is working on this trial, and is authorized to assign, to a physician of its choice, the patient ID number involved in the physician's request.

It is generally desirable for the key manager to be almost stateless, i.e., to remember very little information. He needs to know the trial organizers and to have a current revocation list (in case an organizer, site, or physician has lost some rights), but he should not have a large database. The less information the key manager keeps, the less the cloud's customers will worry about their information being compromised by bribed technicians, court subpoenas, and similar disasters. Furthermore, the scalability of the system is improved by keeping the key manager's state (including his cache) small.

Thus, our scenario features four of the principals involved in a particular clinical trial called `Trial1`:

- `Org1`, the contract research organization conducting `Trial1`.
- `Site1`, one of the sites hired by `Org1` for `Trial1`.
- `Phys1`, one of the physicians working at `Site1` on `Trial1`.
- `KeyManager` for `Trial1`.

We shall discuss and formalize in DKAL the policies that these principals should follow so that, when an authorized physician wants to get a patient record, he gets a key for it, but unauthorized principals don't get keys. The key manager, who has all the keys, counts as authorized.

Informally, the essential parts of these policies are as follows. `Org1` sends, to each site selected for `Trial1`, a (cryptographically signed) certification that it has been hired to work on `Trial1` and has been allocated a certain range of patient ID numbers. Furthermore, `Org1` sends each of these sites a signed statement delegating to the site the authority to grant access to patient records whose ID numbers are in that range. `Site1`, in turn, having received its range `[N1,N2]` of patient ID numbers, sends each of its chosen physicians a signed authorization to access records with ID numbers in a certain sub-interval of `[N1,N2]`. Furthermore, `Site1` forwards to its physicians, including `Phys1`, the statement from `Org1` delegating this access authority to `Site1`. When `Phys1` needs to see the record of a patient whose ID number `N` is in `Phys1`'s assigned interval `[P1,P2]`, he sends a request to `KeyManager`, accompanied by the statements from `Org1` and `Site1` granting that authority, plus verifications of the arithmetical facts that the patient ID is in the range assigned to him by `Site1` (so that Site1's authorization applies) and in the range assigned to `Site1` by `Org1` (so that `Org1`'s delegation of authority applies), plus a proof that these arithmetical facts and the two signed statements establish that he has authority, from `Org1`, to access the record in question.

Finally, `KeyManager`'s policy includes verifying proofs sent to him as well as signatures on documents, trusting `Org1` on such authorizations, and sending the correct key to the requester. (For details, see Section 7.)

3 The World of One DKAL Principal

The part of the world seen by a particular principal can be described as follows. First, there are various facts, which we regard as being recorded in some database(s). In what follows, when we refer to databases, we mean collections of relations, not the additional machinery, such as search and query mechanisms, associated with full-scale databases. For brevity, we also stretch the term "database" to include temporary storage. For example, when a physician receives a key from the key manager, he should not, in view of security considerations, record the key in anything as permanent as what is usually called a database. Rather, he should keep a temporary record of it for as long as it takes him to use the key; then he should destroy it. Since this sort of temporary storage will

play the same role as databases in our discussion, and since we don't want to repeatedly write "databases or temporary storage," we let "databases" refer to both of these.

A principal's data are relations in these databases plus some functions, e.g. standard arithmetical operations, given by algorithms. Some of the relations and functions may be public, with the same contents for all principals. Others may be private for a specific principal. Intermediate situations are also possible, where several but not all principals share some relations and functions.

These relations form what we call the *substrate* of a principal. The public part of a substrate would ordinarily also include things generally available in computing systems, such as numbers and the arithmetical operations on them. In our clinical trial scenario, the publicly available substrate would tell, for example, which contract research organizations are conducting which trials. A contract research organization's substrate would tell which sites are to participate in which trials as well as which patient ID numbers have been allocated to each site. The sites, in turn, would have in their substrates information about the physicians they hired and the patient ID numbers assigned to each physician.

A principal can, in general, obtain and update values of functions in his substrate, but this ability may be limited in several ways. In the first place, some entries may be undefined for the principal — for example, record keys that have not been given to him. Let us suppose that principals use a substrate function `Key` for temporary storage of keys. Then this function would ordinarily be defined at only a small number of arguments in any particular principal's substrate (except in the key manager's substrate).

Furthermore, some parts of the substrate, in particular public parts, may be designated as static, i.e., not susceptible to updating

Another part of a principal's world, the part directly concerned with knowledge and communication, is the *infostrate*. It contains the principal's *knowledge assertions*, which incorporate the knowledge initially available to the principal and the knowledge obtained from communication. In addition, it contains knowledge that the principal has deduced from these together with facts from the substrate. (Actually, there is more to the infostrate, including the principal's policies for sending and accepting communications; see Section 6 for a complete description.)

As we shall see below, the mathematical framework of DKAL treats a principal's substrate and infostrate rather differently. Nevertheless, in real-world situations, there may be some arbitrariness in deciding where a particular aspect of a principal's world should reside. Communications are definitely in the infostrate, and public databases are definitely in the substrate, but between these there are less definite situations. Intuitively, if a principal learns some information from a communication and soon uses it for further communication, this would belong in the infostrate, but if the information is recorded, perhaps in a table, for later use, then the table belongs in the substrate. Ultimately, though, decisions about what goes into the substrate and what goes into the infostrate will be governed

by pragmatic considerations including ease of analysis (which may, of course, benefit from conformity to intuition).

3.1 Substrate Functions and Infon Relations

We adopt several conventions concerning substrates and infostrates.

The substrate of any principal is assumed to be a typed, purely functional, first-order structure, except that the functions may be partial[7]. That is, the substrate is a finite system of nonempty sets together with some (possibly partial) functions between them. Each function has a specified number of arguments, of specified types, and a specified type for its output. One of the types is Boolean, with just two elements, `true` and `false`. Although officially a substrate has only functions, not relations, Boolean-valued functions amount to relations. So in effect we have arbitrary (multi-sorted) first-order structures. Constants are, as usual, regarded as 0-ary functions. Note that, since functions are allowed to be partial, there may be ground terms that are not assigned any value by the usual definition.

A participant's infostrate, in contrast, does not involve function symbols but resembles a relational structure, with the same underlying sets (of various types) as the substrate. Instead of relations, though, it has *infon relations* that assign *infons* (rather than truth values) to tuples of elements of the appropriate types. *Infon formulas* are built from infon relations, constants, and variables according to rules that will be spelled out in Section 5. These infon formulas do not have truth values in the usual sense; rather, their values are *infons*, which we think of as items of information. They are not true or false, as in classical logic, but are known or not known by principals. Infons are the entities that can be communicated from one principal to another. Each principal in a DKAL system has a *policy*, including

- what sorts of communications to accept, and from whom,
- what communications to send, and to whom, and
- what changes to make in its state.

All of these specifications can use information from this principal's substrate and from the principal's *knowledge* and *roster*[8].

The knowledge mentioned here arises from the principal's original knowledge assertions, the communications that the principal has received and accepted, and information from the substrate.

[7] For readers familiar with the abstract state machine literature, we point out the following difference. Suppose that f is a partial function undefined at an argument $\bar{a}$. The abstract state machine convention is that $f(\bar{a}) =$ `undef` where `undef` is a first-class element, so that e.g. $f(\bar{a}) = f(\bar{a})$ evaluates to `true`. In a DKAL substrate there is no value for $f(\bar{a})$, nor for any term that includes it as a subterm. In particular, the value of the equation $f(\bar{a}) = f(\bar{a})$ is undefined.

[8] The roster is actually part of the substrate, but we mention it here for emphasis.

The roster consists of the elements of the substrate that the principal is aware of. The official definitions of "knowledge" and "roster" will be given in Sections 4 and 6.

In practice, it is difficult to separate "knowledge" and "policy" as described above. For example, if a principal decides to trust another on some infon (i.e., "if so-and-so tells me x then x") is this trust assertion a matter of knowledge or of policy? Furthermore, it has become traditional in logic-based authorization to use the word "policy" in a broad sense that includes some of what we have called knowledge, at least the explicit knowledge assertions of a principal. Accordingly, we shall from now on use "policy" in this broader sense. In particular, the formalization of our example in Section 7 will include, in the principals' policies, several trust assertions.

A crucial aspect of infons (or, more precisely, of the infon formulas that denote them) is that, in addition to atomic infon formulas (asserting an infon relation of some named substrate elements) and combinations via "and" and "implies", they can also assert that some principal said (or implied) some other infon. In this way, communications from one principal can enter into the knowledge of another.

3.2 Notational Conventions

In this paper, we choose our notations for function names, infon relations, and variables in accordance with the following conventions. All of these symbols are in the typewriter typeface.

Function symbol: String of letters and numbers, beginning with a capital letter and containing at least one lower-case letter.

Infon relation: One or more strings of letters, each string starting with a lower-case letter. (The "or more" is to allow writing some arguments between the parts of the infon relation.)

Variable: Nonempty strings of capital letters, possibly followed by digits.

In addition, we use obvious notation for arithmetical operations and propositional connectives.

Here are some examples. Variables in our examples will include `TRIAL`, `SITE`, `N1`, and `PERSON`.

Function names will include constants (recall that these are nullary functions, i.e., taking no arguments), such as `Trial1`, `Org1`, and `Phys1`. They also include unary functions, such as `Org` (as in `Org(TRIAL)`) and `Key` (as in `Key(RECORD)`), and functions of two or more arguments like `Record` (as in `Record(N, TRIAL)`).

Infon relations include the binary `participates in` and ternary `participates in at as physician`, as in `SITE participates in TRIAL` and `PERSON participates in TRIAL at SITE as physician`.

There is a special infon relation, `asinfon`, which takes one argument of Boolean type and returns an infon whose intended meaning is that the argument is true. That is, it converts classical truth values (things that can be true

or false) into infons (things that can be known by principals). An example, using the availability of elementary arithmetic and propositional logic in the substrate, is `asinfon(N1` $\leq$ `N` **and** `N` $\leq$ `N2)`.

3.3 Disclaimer

Our primary goal in this paper is to describe the communication and inference mechanisms of DKAL. That is, we shall be interested primarily in infostrate phenomena. To better expose these essential ingredients of DKAL, we shall omit from our examples various things that, though important in reality, would only clutter the example and obscure the main points we wish to make. In particular, we shall not deal here with the updates that principals may perform in their substrates. (Some special updates of substrates, namely additions to principals' rosters, occur automatically as an effect of communication; these are included in the discussion in Section 6.)

Principals' policies may evolve. For example, companies may have to change their policies (keeping a log, of course, to show what their policy used to be) when a government agency changes its rules. They may have to delete or modify some of their knowledge assertions, acquire new knowledge assertions, amend some of their communication rules, etc. We postpone to a future treatment the issue of policy evolution, except that we define, in Subsection 6.3, how accepted communications become new knowledge assertions. For the sake of simplicity, we do not treat in this paper deletions or modifications of knowledge or removal of elements from rosters. Nor do we treat explicit additions, deletions or modifications of communication rules or filters.

Furthermore, although DKAL is a typed system, whose types always include the types of Boolean values, of integers, and of principals, and usually include many other application-specific types, we shall omit explicit indications of types in our examples, trusting in the reader's common sense to correctly understand the types of the expressions we write.

4 Substrate

In logic terms, the substrate of a principal is a multi-type (also called multi-sorted) first-order structure where relations are viewed as Boolean-valued functions. Accordingly the Boolean type, comprising two elements `true` and `false`, is ever present. Other ever present types are the types of principals and integers. The substrate vocabulary contains the names of the substrate functions. Constants are (the names of) nullary functions. Terms over the substrate vocabulary are produced as usual (in first-order logic) from variables by means of (repeated applications of) function names. In particular, constants are terms.

It is convenient to see the substrate as a collection of (types and) partial functions. For instance, the substrate of principal `Org1` contains a binary function `Status(SITE,TRIAL)` one of whose values is `Unnotified`. The intended meaning of `Unnotified` is "chosen to work on this trial but not yet notified". Another

binary function in the substrate of `Org1` is `Patients(SITE,TRIAL)`, which gives the range of patients' identification numbers assigned to the `SITE` at the `TRIAL`.

Some substrate functions, like `Status` or `Patients`, are finite and can be given by tables. In general, a substrate function f is given by an algorithm (which can be just a table lookup). For example, integer multiplication is given by an algorithm and so is integer division. We presume that, given a tuple $\bar{a}$ of arguments of appropriate types, the algorithm first decides whether f is defined at $\bar{a}$. If $f(\bar{a})$ is undefined, the algorithm reports that fact and halts. Otherwise it proceeds to compute $f(\bar{a})$. Thus the algorithm never "hangs". For example, given $(1, 0)$, the algorithm for integer division will report that $1/0$ is undefined.

Equivalently one can think of a substrate as a relational database. Total Boolean-valued functions are relations. Any other function f, of arity j, gives rise to a $(j+1)$-ary relation (the graph of f) with a functional dependency that says that the first j columns determine the last column. For example, the binary function `Status(SITE,TRIAL)` gives rise to a ternary relation, with attributes for sites, trials and statuses, where the values of `SITE` and `TRIAL` determine the value of `STATUS`.

4.1 Canonic Names, and Term Evaluation

We presume that every element in the substrate of a principal A has a canonic name that is a ground term. We identify these elements with their canonic names. It is presumed that canonic names are obviously recognizable as such. For example, A's canonic name for a principal P could be a record identifying P, e.g. via his public cryptographic key, and giving his address. The canonic name for an integer could be its binary notation.

> **Quisani**[9]**:** What if the substrates of two principals A and B share an element? Can the two names differ? For example, is it possible that A uses binary notation for integers while B uses decimal?
>
> **Authors:** No. We presume names have been standardized for common substrate elements.
>
> **Q:** I see a possible problem. Elements of different types might have the same canonic name. For example, if you had identified principals by just their cryptographic keys, those could also be the binary notations for integers.
>
> **A:** Canonic names are ground terms and thus have definite types.

To evaluate a ground term t at a state S means to produce the canonic name for the element $[\![t]\!]_S$ denoted by term t at S. The principal may be unable to evaluate a ground term. For example, in a physician's substrate the function `Key(RECORD)` is defined for few, if any, records. For a particular record, say

[9] Quisani is an inquisitive friend of ours.

`Record`(10,`Trial1`) of patient 10 in `Trial1`, the physician may be unable to evaluate the term `Key`(`Record`(10,`Trial1`)). The values of `Key` that a physician can evaluate, i.e., the ones that are defined in his substrate, are (in our scenario) the ones which he has gotten from the key manager (and has recorded and not yet destroyed).

4.2 Roster

Intuitively the *roster* of a principal A is the finite collection of elements known to principal A. The roster splits into *subrosters*, one subroster for every type. The subroster of type T consists of elements of type T. Formally, the subrosters are unary relations.

Unsurprisingly, the Boolean subroster consists of `true` and `false`, and the principal subroster of A contains at least (the canonic name of) A.

Roster may not be closed under substrate functions. For example, even if the substrate has a principal-to-principal function Manager, the principal subroster may contain some employee `John` but not contain (the canonic name for) `Manager(John)`. In particular the integer subroster contains only finitely many integers.

5 Logic

5.1 Infons

Traditionally, in mathematical logic, declarative statements represent truth values. In DKAL, as explained in Section 3, we view declarative statements as containers of information and call them infons [12]. This has a relativistic aspect. Indeed, the world of DKAL consists of communicating principals, and each principal lives in his own state. As a result, one question, whether a given statement φ is absolutely true or absolutely false, is replaced with as many questions as there are principals: Does a given principal know φ?

In the first part of the present section, we fill in details about infons, making precise what was outlined in Section 3.

Recall the connection between truth values and infons, given by the `asinfon` construct.

> **Q:** I guess a principal A knows `asinfon`(b) if and only if the Boolean value b is true.
>
> **A:** That is the idea but one has to be a bit careful. It may be infeasible for the principal A to carry out the evaluation. For example,
>
> $$3^{3^{3^3}} + 8^{8^{8^8}} \texttt{ is prime}$$
>
> has a well defined Boolean value but A may not know it.

5.2 Infons as formulas

Traditionally, statements are represented by formulas. In the previous DKAL papers, we treated infons as objects and represented infons by terms; in particular we had infon-valued functions. In the meantime we realized that infons can be naturally represented by formulas, and that is what we do in this paper.

> **Q:** Since you are interested in statements like "a principal A knows infon formula φ", you may want to work in epistemic logic where formula $K_A(\varphi)$ means exactly that A knows φ.
>
> **A:** It is convenient to have the knowledge operator K_A implicit when you work in the space of A.
>
> **Q:** Explicit knowledge operators allow you to nest them, e.g. $K_A K_B \varphi$.
>
> **A:** Allows or provokes? In the world of DKAL, A never knows what B knows, though A may know what B said.

Substrate Vocabulary Recall that a principal's substrate is a many-typed structure of sets (interpretations of type symbols) and partial functions (interpretations of function symbols). We assume that the number of these symbols is finite. Every function symbol f has a type of the form $T_1 \times \ldots \times T_j \to T_0$ where j is the arity of f and the T_i are type symbols. Constants are nullary function symbols. The types of Boolean values, integers and principals are always present. Recall that relations, e.g. the order relation $\leq$ on integers, are treated as Boolean-valued functions.

For every type, there is an infinite supply of variables of that type. We use numerals and standard arithmetical notation. In addition we use strings for the names of variables and functions; see Section 3 for our conventions concerning these strings and for examples of their use.

Infon relations Traditionally, relations take values `true` and `false`. *Infon relations* are similar except that their values are infons. Their arguments come from the substrate of a principal. Each infon relation has a particular type for each of its argument places. So infon relations assign infons to tuples of elements of the appropriate types.

Atomic infon formulas Prefix or infix notation is used for infon relations. Here are some examples.

```
SITE participates in TRIAL
SITE is allocated patients N1 to N2 in TRIAL
asinfon(N1 ≤ N and N ≤ N2)
```

Composite infon formulas These are built from atomic formulas by means of conjunction $\wedge$, implication $\rightarrow$ and two unary connectives p `said` and p `implied` for every term p of type principal. To make complicated implications easier to read, $\alpha \rightarrow \beta$ can be written as

if
α
then
β

Here's an example:

if
`asinfon(N1` $\leq$ `N` **and** `N` $\leq$ `N2)` **and**
`SITE implied PERSON may read Record(N,TRIAL)`
then
`Org1 implied PERSON may read Record(N,TRIAL)`

Syntactic sugar Implications

$(p$ `said` $\varphi) \rightarrow \varphi$,
$(p$ `implied` $\varphi) \rightarrow \varphi$

are abbreviated to

p `is trusted on saying` φ,
p `is trusted on implying` φ.

In fact, these abbreviations are so common that they are often further abbreviated to p `tdonS` φ and p `tdonI` φ respectively. We note that a similar abbreviation, `A controls s` for `(A says s)` $\rightarrow$ `s`, was used in [1, page 7].

5.3 Logics and Theories

In [13], two of us introduced and investigated propositional infon logics, in particular *primal* infon logic where the derivation problem, whether a given formula φ follows from a given set Γ of formulas, is solvable in linear time. Primal infon logic with (universally quantified) variables is obtained from propositional primal logic by adding the substitution rule. The derivation problem for primal logic with variables remains feasible (though not linear time) [6,7]. Furthermore, security policies involve arithmetic, etc. Instead of pure infon logic, we should be working with infon theories.

Separation of concerns The previous versions of DKAL had a built-in version of infon logic. Here we work with infon theories, and — to separate concerns — we view infon theory as a parameter. In the rest of the article:

– infon logic is primal logic with variables or some extension of it that is a sublogic of classical first-order logic, and

– infon theory includes the theory of arithmetic in the infon logic.

Infon theory is to be given by some axioms. There is considerable flexibility in the choice of axioms. In particular there is no need to minimize the system of axioms. In fact, for our purposes a rich system of axioms may be more appropriate. It is, however, necessary that the axioms be efficiently recognizable.

When we say that a set Γ of infon formulas entails an infon formula φ we mean that Γ entails φ in the fixed infon theory.

From time to time, a DKAL principal may need to compose a proof in the infon theory. In difficult cases he may use an automated or interactive prover to compose proofs. In any case, the methods used for proof composition, even if formally specified, belong to a level of abstraction lower than what we consider in this paper.

Remark 1. The current implementation of DKAL [9] employs primal infon logic with variables and uses the SQL database engine. The infon theory uses ground arithmetical facts that are checked by means of the SQL engine.

Remark 2. In this and the previous subsections, we speak about infon theory used by principals. A more powerful theory may be needed for the analysis of security policies and their interaction.

5.4 Justifications

1. If φ is an infon formula in one of the following two forms
   1a. `A said` α`,`
   1b. $\beta \rightarrow$ `A implied` α,
   then a cryptographic signature of principal `A` under (a strong hash of) the whole formula φ, accompanied with the canonic name of `A`, is a *justification* for φ. The signature is specific for (the hash of) φ; thus changing the formula would (with overwhelming probability) make the signature invalid[10].
2. Let φ be an arbitrary infon formula. A *justification* for φ is a derivation of φ in the fixed infon theory from hypotheses of the form (a) or (b) together with justifications by signatures for all the hypotheses[11].

Q: You mention a cryptographic signature under a formula φ. Do you mean that φ needs to be encrypted?

A: Not necessarily. This depends on the purpose and may vary even within one application.

Q: In connection with case 1b, what if A wants β to be checked by a particular principal `B`, essentially delegating the matter to B?

[10] In case 1a, our notion of justification is similar to Binder's notion of certificate [8].

[11] Proof-carrying communications, in higher-order logic, were used by Appel and Felten [3]. In contrast, the current implementation [9] of DKAL uses logic that is decidable and in fact feasible.

A: He may replace β with `(B said` β`)` or with `(B implied` β`)` depending on whether he intends to allow `B` to delegate the matter further.

Q: What about `asinfon` formulas? How can they be justified?

A: Only by a derivation in the infon theory, as in item 2 above. Some of them may happen to be axioms.

Q: So many `asinfon` formulas may not be justifiable.

A: That's right. First, the substrates of different principles may disagree. Second, even in the case when only public relations are involved, where all principals agree, things might change over time. But the infon theory should be global (for all principals) and permanent (for all time).

6 Infostrate

The infostrate of a principal A is determined by the substrate and these:

- Knowledge assertions,
- Communication rules,
- Filters.

6.1 Knowledge

At each state, each principal has a finite set of knowledge assertions, which are infon formulas in the vocabulary of the principal enriched by his roster. This set is composed of original knowledge assertions that the principal always had and other knowledge assertions that he acquired from communications. In our example, `Site1` has an original knowledge assertion

```
Org(TRIAL) is trusted on saying SITE participates in TRIAL
```

and he learns

```
Org1 said Site1 participates in Trial1
```

A knowledge assertion of a principal may have free variables. Its instances in a particular state are obtained by replacing variables with the principal's roster elements of the corresponding types.

Knowledge assertions constitute one source of the principal's knowledge. Another source is his substrate. If the principal successfully evaluates to `true` a ground substrate term t of Boolean type then he learns `asinfon`(t). Further, the principal may use the fixed infon theory to derive additional infon formulas from his knowledge assertions and substrate facts. We shall say that a principal *knows* an infon formula φ at a state S if there is a feasible derivation of φ in the infon theory from that principal's knowledge assertions and substrate information.

Q: This "feasible" is vague. If it takes the principal two minutes to deduce φ, fine. But if φ happens to be P=NP and if it happens to be deducible with a few years' effort, must he deduce that?

A: Well, there may be situations where even two minutes is too long a time. But your point is well taken. The decision how much work and time to put into deducing φ depends on the scenario at hand.

6.2 Communication

The infostrate of a principal A may (and typically does) contain communication rules. These rules may cause A to send communications to various principals. We presume in this paper that communications automatically come with identifications of the senders. The recipients filter their communications; a communication may be accepted or rejected.

Communications of one principal to another may or may not be accompanied by a justification of their contents. In the future, a communication accompanied by a justification of its content will be called *justified*. As we already said in the introduction, both kinds of communications are important but, in this paper, we restrict attention to justified communication.

The work in this paper is independent of the mechanisms by which principals communicate. The mechanisms could include message passing, shared memory, bulletin boards, etc. In particular, even when one principal sends a communication to just one other principal, we do not presuppose that it is necessarily message passing.

The communication rules of any principal A have two possible forms:

(Send) **if** π
then send [**justified**] **to** r
φ

and

(Post) **if** π
then post [**justified**] **to** x
φ

Here π and φ are infon formulas, r is a principal term, and x is a variable of type principal not occurring in π. We call π the *premise*; both r and x are *recipient* terms; and φ is the *content* of the rule. Further, **justified** is optional.

Q: What's the difference between sending and posting, apart from the restriction, in posting, that x is a variable not in the premise?

A: Intuitively, posting does not require the sender to know all the recipients. Formally, the difference will become clear when we talk about how to instantiate variables.

The variables that occur in a communication rule split into two categories. The variables that occur in the premise or in the recipient term are intended to be instantiated by principal A, the sender, before he sends anything out. In the case of (Send), all these variables are instantiated by elements of A's roster. In the case of (Post), the variables in the premise π are instantiated by elements of A's roster but the recipient variable x is thought of as instantiated by all elements of its type in A's substrate.

> **Q:** Now the difference is clear, but what is this "thought of"? That seems like an evasion. I think I see the problems that you try to evade. One is that there could be a huge number of elements of the type of x. Another is that, even if there are only a few elements, A may not know them all.
>
> **A:** You are right. These are the problems. We don't intend for A to actually send individual messages to all these recipients, whether there are many of them or few. Rather, he should post the content φ where the intended recipients can read it or download it, etc.

For simplicity, in the rest of the paper, we consider only (Send) rules, always justified. Our example in Section 7 will have only such rules.

The variables of a communication rule that occur only in the content are called *verbatim* variables. They are sent out uninstantiated, verbatim, and will be instantiated by the receivers using elements of their rosters.

> **Q:** Obviously a variable v of type T is instantiated by elements of type T. Suppose that v is one of those verbatim variables. Clearly the sender has the type T; otherwise he would never have such a variable. What about the receiver? He also needs T in order to instantiate v properly. How does the sender know that the receiver has T?
>
> **A:** The language does not guarantee that the sender knows this. If the receiver does not have T, the communication is unintelligible to him and therefore rejected.
>
> **Q:** OK, let us assume that the sender and receiver both have type T in their vocabularies. Should it be the same type in the semantic sense; that is, should T have the same elements in both substrates?
>
> **A:** Not necessarily. Consider the type Secretary. The secretaries on the sender side and those on the receiver side may be different but the sender may consciously have in mind the receiver's secretaries.
>
> **Q:** What if the receiver has type T but it is not at all the type that the sender had in mind? For example, being the government, the receiver thinks that "secretary" means a member of the president's cabinet.
>
> **A:** Too bad. This communication will be misunderstood.
>
> **Q:** OK, this clarifies the situation for types. Something else bothers me. Suppose the content contains $f(v)$ where v is a verbatim variable or,

more generally, $f(t)$ where t contains a verbatim variable. The sender can't evaluate this, so the recipient must. How does he interpret the symbol f?

A: The situation for such function symbols is the same as for types of verbatim variables. The recipient must have f in his vocabulary, with the same type as the sender's f; otherwise the communication is unintelligible. And if the recipient's interpretation of f is seriously different from what the sender intended, then the communication may be misunderstood.

In a state S of principal A, the communication rule (Send) may result in a number of communications to various principals. A *sender* substitution ζ for (Send) instantiates the variables in the premise π and the recipient term r (that is, all non-verbatim variables) with canonic names from the current roster. If A can evaluate the infon formula $\zeta(\pi)$ and knows the resulting infon, and if he can evaluate $\zeta(r)$, then he applies ζ to the content φ and evaluates the ground terms in the resulting infon formula. If the evaluation fails because of an undefined term, that instance of (Send) produces no communication. Suppose the evaluation succeeds, and let's call the resulting infon formula $\zeta(\varphi)$. Thus $\zeta(\varphi)$ is obtained from φ by substitution followed by evaluation. If A is able to construct a justification for $\zeta(\varphi)$ then communication $\zeta(\varphi)$, complete with a justification, is sent to principal $\zeta(r)$.

Q: This "able to construct" sounds vague to me.

A: You are right. It is vague in the same sense as "feasible". The effort devoted to constructing justifications would depend on the scenario at hand.

Q: OK. But now what if the sender has two (or more) justifications for the content φ? Which one does he send?

A: DKAL imposes no restrictions on the choice of justification.

For some applications, we would want to allow, in the content φ of (Send), *verbatim function symbols*. These would not be evaluated during the computation of $\zeta(\varphi)$, even if all their arguments have been successfully evaluated. Rather they would be sent as is, to be evaluated later. For example, an executive A tells another executive B: "Have your secretary call my secretary to set up an appointment". The first occurrence of "secretary" needs to be verbatim if A does not know B's secretary. The second occurrence of "secretary" should not be verbatim. Verbatim function symbols are introduced in [14]. We will not use them in our example in Section 7. Note, however, that our communication rules do contain occurrences of function symbols that the sender will not evaluate, namely ones whose arguments cannot be evaluated because they involve verbatim variables.

Speeches A send rule of principal A of the form

if π
then send [justified] to r
A `said` ψ

may be abbreviated to

if π
then say [justified] to r
ψ

6.3 Filters

A principal filters incoming communications. He may have several filters, and a communication will be accepted provided at least one of the filters admits it. In the world of Evidential DKAL where all communications are justified, a filter has the form

(Filt) **if** ρ
then accept if justified from s
Ψ

Here ρ is an infon formula, called the *premise*, and s is a principal term, called the *sender* term. Further, Ψ is a pattern, which is defined as follows. An *atomic pattern* is an atomic infon formula or an infon variable. *Patterns* are either atomic patterns or built from them exactly as composite infon formulas are built from atomic infon formulas, that is, by means of conjunction, implication, p `said`, and p `implied`.

The part "if justified" in (Filt) means that the communication must be accompanied by a justification which the recipient should verify before accepting the communication.

Suppose principal B receives a communication φ from another principal A. First, B adds (the canonic name of) A to his roster, unless it is already there. (B may want to remove A from the roster later on, but he should have A on the roster at least while processing φ.) Second, for each of his filters F, the principal B tries instantiating the variables in the premise ρ and the sender term s, using elements of his roster. He wants an instantiation η such that he knows $\eta(\rho)$ and such that $\eta(s)$ evaluates to A. For each instantiation η that succeeds in this sense, he checks whether the communication φ matches the pattern $\eta(\Psi)$ in the following sense. He can get φ from $\eta(\Psi)$ by replacing infon variables with infon formulas and replacing the remaining variables with terms of the same types. Third, for each instantiation η that has succeeded so far, B verifies the justification of φ. If the verification also succeeds, then he adds φ to his knowledge assertions, and he adds to his roster all the canonic names that he can obtain by evaluating ground terms that occur in φ.

Q: Is verifying a justification also a vague notion?

A: No, the recipient must really verify the justification. Notice though that the work required is bounded in terms of the size of the communication φ plus the justification.

Q: Right. In fact, it seems to be bounded just in terms of the justification.

A: Not quite. Recall that the justification could be a signature on a hash of φ. Verifying it would require computing the hash of φ from φ.

Q: You showed how filters look in Evidential DKAL. I guess that in general not-necessarily-evidential DKAL they look the same except that **if justified** may be omitted and then the recipient does not have to check a justification.

A: Not quite. There is an additional difference. The content φ does not automatically become a knowledge assertion of the recipient. The filter may give other ways to accept a communication. For example, the 911 operators should not automatically believe every anonymous report they get, but they should not completely ignore a report either.

In the present paper, for brevity, we make the following simplifying assumption. Each principal has exactly one filter, namely

if `asinfon(true)`
then accept if justified from x
  Ψ

where x is a principal variable and Ψ is an infon variable. In other words, all communications are accepted subject to the verification of justification.

7 Clinical Trial Scenario: Policies

The clinical trial scenario was described informally in Section 2. Here we formulate the relevant parts of the policies of `Org1`, `Site1`, `Phys1` and `KeyManager` in DKAL. For brevity, we do not declare the types of the variables, leaving it to the reader's common sense to interpret variables correctly. We also remind the reader that we use "record" to mean the part of a patient's medical record that should be accessible to his trial physician; `Record(N, TRIAL)` refers to record, in this sense, of the patient with ID number `N`.

7.1 Policy of Org1

if
   `asinfon(Org(TRIAL) = Org1)`
   `asinfon(SiteStatus(SITE,TRIAL) = Unnotified)`
   `asinfon(SitePatients(SITE,TRIAL) = [N1,N2])`
then
   say justified to `SITE`
      `SITE participates in TRIAL`
      `SITE is allocated patients N1 to N2 in TRIAL`
   send justified to `SITE`
      if
         `asinfon(N1` $\leq$ `N` **and** `N` $\leq$ `N2)`
         `SITE implied PERSON may read Record(N,TRIAL)`
      then
         `Org1 implied PERSON may read Record(N,TRIAL)`

Q: You surely intend that the sites, chosen by `Org1` to participate in a particular trial, are notified about that just once. But what prevents `Org1` from executing the rule over and over again?

A: Currently the rule has one **say** and one **send** commands. To make our intention explicit, we can add a third command, namely an assignment that changes the value of `SiteStatus(SITE,TRIAL)`.

Q: Changes it to `Notified`? Or maybe the new value should be `Pending`, which later is changed again to reflect the site's reaction.

A: That's the thing. Communication rules may have side effects, in the form of substrate updates, which detailed policies would specify. That would involve decisions that are irrelevant to us here. Accordingly, we omit in this example the explicit updates of the substrate[12]. Concerning `Org1`, we just assume that, for each clinical trial and each site chosen by `Org1` to participate in the trial, the communication rule is executed exactly once.

Q: Concerning the syntax of the rule, I see that the two keyword pairs **if . . . then** are used for two different purposes.

A: Yes, one **if** starts the premise of the rule, and the corresponding **then** starts the body. Another pair is syntactic sugar in an infon formula; it is used to render an implication in a form that is easier to read. As long as we're talking syntax, let's also clarify another point. When two or more infon formulas are exhibited, one after the other without connectives between them, we mean the conjunction. The premise above is the conjunction of three infon formulas. The content of the **say** is the

[12] Related issues are addressed in [4].

conjunction of two infon formulas, and the premise of the implication is the conjunction of another pair infon formulas.

Q: Now let me look into the necessary justifications. Isn't it true that all that is needed is two signatures of `Org1`?

A: Yes. One signature goes with the **say justified** command, and the other with the **send justified** command. In the second case, a signature suffices because the content has the form $\alpha \rightarrow$ `Org1 implied` β.

7.2 Policy of Site1

```
Org(TRIAL) is trusted on saying
   SITE participates in TRIAL
Org(TRIAL) is trusted on saying
   SITE is allocated patients N1 to N2 in TRIAL

if
   Site1 participates in TRIAL
   Site1 is allocated patients N1 to N2 in TRIAL
   asinfon(PhysStatus(PHYS,Site1,TRIAL) = Unnotified)
   asinfon(PhysPatients(PHYS,Site1,TRIAL) = [P1,P2])
   asinfon(N1 ≤ P1 and P2 ≤ N2)
then
   say justified to PHYS
      PHYS participates in TRIAL at Site1 as physician
      PHYS is allocated patients P1 to P2 in TRIAL at Site1
   send justified to PHYS
      if asinfon(P1 ≤ N and N ≤ P2)
      then Site1 implied PHYS may read Record(N,TRIAL)
   send justified to PHYS
      if
         asinfon(N1 ≤ N and N ≤ N2)
         Site1 implied PERSON may read Record(N,TRIAL)
      then
         Org(TRIAL) implied PERSON may read Record(N,TRIAL)
```

Q: It is presumed I guess that, for each trial that `Site1` participates in, `Site1` chooses appropriate physicians and then notifies them just once.

A: Exactly.

Q: Now again let me look into the three justifications. The first two are simply `Site1`'s signatures. What about the third?

A: Here `Site1` is simply forwarding, literally, what it received from `Org1`, with the original signature of `Org1`. That's it.

Q: Literally? One discrepancy is that there was a variable `SITE` in `Org1`'s communication rule. I do not see it in the forwarded version.

A: That variable occurred in the premise of `Org1`'s rule. So it was instantiated. The actual communication from `Org1` to `Site1` has `Site1` in its place.

Q: There is another discrepancy. The last line of `Site1`'s rule has `Org(TRIAL)` where `Org1`'s policy has `Org1`.

A: The variable `TRIAL` occurs also in the premise of `Site1`'s rule, so it will be instantiated. Let's suppose that `Org1` nominated `Site1` to participate in `Trial1` and assigned to it a range `N1` to `N2` of patient IDs. We presume that function `Org` is public. In particular, `Site1` can evaluate `Org(Trial1)` to `Org1`. Accordingly, it trusts `Org1` on saying that `Site1` participates in `Trial1` and is allocated the range `N1` to `N2`. So, in its communication rule, `Site1` can instantiate `TRIAL` to `Trial1`. Then `Org(TRIAL)` that you asked about will be instantiated to `Org(Trial1)` and will be evaluated to `Org1`.

7.3 Policy of Phys1

```
SITE is trusted on saying
   PHYS participates in TRIAL at SITE as physician
SITE is trusted on saying
   PHYS is allocated patients P1 to P2 in TRIAL at SITE

KeyManager is trusted on saying
   key of Record(N,TRIAL) is K


if
   Phys1 participates in TRIAL at SITE as a physician
   Phys1 is allocated patients P1 to P2 in TRIAL at SITE
   asinfon(P1 ≤ N and N ≤ P2)
   asinfon(NeedInfo(N))
then
   send justified to KeyManager
      Phys1 said Phys1 requests to read Record(N,TRIAL)
      Org(TRIAL) implied Phys1 may read Record(N,TRIAL)
```

Q: I understand that `Phys1` sends to `KeyManager` the conjunction of two infon formulas. For the first conjunct, you could have used the **say** command with the content `Phys1 requests to read Record(N,TRIAL)`. Wouldn't that look more natural?

A: Maybe, but it would require separating the two conjuncts. If the implementation then sends them separately, the `KeyManager` would have to remember one conjunct until he gets the other.

Q: All right. Now let's look at the justification. The variables `N` and `TRIAL` are instantiated, say to 10 and `Trial1`, and of course `Org(TRIAL)` then evaluates to `Org1`. `Phys1` needs to justify a conjunction which he can get by justifying both conjuncts. The first conjunct is easy. It can be justified by `Phys1`'s signature. How does he justify the second conjunct?

A: He needs to deduce the second conjunct in the fixed infon theory from the following infon formulas. First there is `Org1`'s signed statement

```
(1)   if      asinfon(N1 ≤ N and N ≤ N2)
              SITE implied PERSON may read Record(N,TRIAL)
      then    Org1 implied PERSON may read Record(N,TRIAL)
```

Second there is `Site1`'s signed statement

```
(2)   if      asinfon(P1 ≤ N and N ≤ P2)
      then    Site1 implied PHYS may read Record(N,TRIAL)
```

Finally, there are the arithmetical facts (with no variables!)

```
(3)   asinfon(N1 ≤ 10 and 10 ≤ N2)
      asinfon(P1 ≤ 10 and 10 ≤ P2)
```

for which `Phys1` can easily supply proofs in the infon theory.

Q: Aren't `N1, N2, P1, P2` variables?

A: They have been instantiated but, by slight abuse of notation, we continue to use the same symbols.

Q: In order to combine (1)–(3), I guess, `Phys1` will need to apply the substitution rule of infon logic: `N` becomes 10, `SITE` becomes `Site1`, `PERSON` and `PHYS` become `Phys1`, and `TRIAL` becomes `Trial1`.

A: Right. And once that's done, all that remains is to apply modus ponens a few times.

7.4 Policy of KeyManager

```
if
   PERSON said PERSON requests to read Record(N,TRIAL)
   Org(TRIAL) implied PERSON may read Record(N,TRIAL)
   asinfon(Key(Record(N,TRIAL)) = K)
then
   say justified to PERSON
      key of Record(N,TRIAL) is K
```

Q: Why require that `KeyManager`'s communication be justified? The person, say `Phys1`, gets the key. It is more important that the key is encrypted.

A: Yes, the key should be encrypted. One can argue that a signature of `KeyManager` under the infon is useful. In any case, we agreed to restrict attention to justified communication in this paper.

Q: It seems artificial that the key manager sends an infon to `Phys1` rather than just the key.

A: It makes sense to indicate what the key is for, especially because `Phys1` might have asked for several keys at once, and the key manager obliged.

8 Future work

It is about time that logic-based authorization graduates from academic labs to industrial applications. There is a great need for specification-level authorization, and many organizations, e.g. the technical committees for TSCP [19] and for XACML [20], understand that. Other potential users of these methods may not yet even be aware of them. Logic-based authorization should be prepared to meet the evolving needs of the customers. The languages should be both sufficiently rich and user-friendly.

We are actively working on a number of extensions of DKAL. Here we mention just three of them.

One pressing issue for DKAL is the need to manage substrates in a high-level transparent way. Essentially the same issue is discussed in [4]. We intend to use the abstract state machine (ASM) approach that has been used for some time at Microsoft [16,18] and elsewhere. We view a principal's substrate as an ASM state. The notion of communication rule should be expanded to a notion of rule that allows, in addition to **send** and **post** commands, arbitrary ASM rules to update the substrate.

Similarly, we need to expand the communication mechanism of DKAL so that it can be used to modify not only a principal's knowledge assertions but also his communication rules and filters (and ASM rules).

And, as we mentioned in the introduction, we also need to bring non-justified communication into the current framework.

References

1. Martín Abadi, Michael Burrows, Butler Lampson, Gordon Plotkin: A Calculus for Access Control in Distributed Systems. ACM Trans. on Programming Languages and Systems, 15:4, 706–734 (1993)
2. Alexandra Y. Aikhenvald: Evidentiality. Oxford University Press (2004)
3. Andrew W. Appel, Edward W. Felten: Proof-Carrying Authentication. In: 6th ACM Conference on Computer and Communications Security, 52–62 (1999)
4. Moritz Y. Becker: Specification and Analysis of Dynamic Authorisation Policies. In: 22nd IEEE Computer Security Foundations Symposium, 203–217 (2009)
5. Moritz Y. Becker, Cédric Fournet, Andrew D. Gordon: SecPAL: Design and Semantics of a Decentralized Authorization Language. Journal of Computer Security 18:4, 597–643 (2010)
6. Lev Beklemishev, Yuri Gurevich: Infon Logic (tentative title). In preparation
7. Andreas Blass, Yuri Gurevich: Hilbertian Deductive Systems and Horn Formulas (tentative title). In preparation
8. John DeTreville: Binder, a Logic-Based Security Language. In: IEEE Symposium on Security and Privacy, 105–113, (2002)
9. DKAL at CodePlex. `http://dkal.codeplex.com/`, viewed July 6, 2010.
10. Shayne Cox Gad (ed.): Clinical Trials Handbook. Wiley (2009)
11. Yuri Gurevich: Evolving Algebra 1993: Lipari Guide. In: Specification and Validation Methods, Oxford University Press, 9–36 (1995)
12. Yuri Gurevich, Itay Neeman: DKAL: Distributed-Knowledge Authorization Language. In: 21st IEEE Computer Security Foundations Symposium, 149–162 (2008)
13. Yuri Gurevich, Itay Neeman: Logic of Infons: The Propositional Case. ACM Trans. on Computational Logic, to appear. See `http://tocl.acm.org/accepted.html`.
14. Yuri Gurevich, Itay Neeman: DKAL 2 — A Simplified and Improved Authorization Language. Microsoft Research Technical Report MSR-TR-2009-11 (2009)
15. Yuri Gurevich, Arnab Roy: Operational Semantics for DKAL: Application and Analysis. In: 6th International Conference on Trust, Privacy and Security in Digital Business, Springer LNCS 5695, 149–158 (2009)
16. Yuri Gurevich, Benjamin Rossman, Wolfram Schulte: Semantic Essence of AsmL. Theoretical Computer Science 343:3, 370–412 (2005)
17. Butler Lampson, Martín Abadi, Michael Burrows, Edward P. Wobber: Authentication in Distributed Systems: Theory and Practice. ACM Trans. on Computer Systems 10:4, 265–310 (1992)
18. Spec Explorer. Development `http://msdn.microsoft.com/en-us/devlabs/ee692301.aspx` and research `http://research.microsoft.com/en-us/projects/specexplorer/`, viewed July 6, 2010
19. TSCP, Transglobal Secure Collaboration Program, `http://tscp.org/`, viewed July 6, 2010
20. XACML, Extensible Access Control Markup Language, `http://xml.coverpages.org/xacml.html`, viewed July 6, 2010