DKAL 2 — A Simplified and Improved Authorization Language

Yuri Gurevich
Microsoft Research Redmond
gurevich@microsoft.com

Itay Neeman
Department of Mathematics, UCLA
ineeman@math.ucla.edu

February 2009

Abstract

Knowledge and information are central notions in DKAL, a logic based authorization language for decentralized systems, the most expressive among such languages in the literature. Pieces of information are called infons. Here we present DKAL 2, a surprisingly simpler version of the language that expresses new important scenarios (in addition to the old ones) and that is built around a natural logic of infons. Trust became definable, and its properties, postulated earlier as DKAL house rules, are now proved. In fact, none of the house rules postulated earlier is now needed. We identify also a most practical fragment of DKAL where the query derivation problem is solved in linear time.

Note (added on May 11, 2009) In the meantime, we made the following notational changes.

1. Function `put` of type Info→Speech is renamed to `implied`.
   The new notation makes it obvious that
   (p `implied` x) is weaker than (p `said` x).

2. The conjunction of infons x, y is denoted as $x \wedge y$ rather than $x + y$. While the latter notation made sense in the original algebraic approach, it is not natural in the logic approach.

Contents

1 Introduction

In the real world, access control is still handled primarily by ACLs, access control lists. ACLs are outdated. The question is what should 21st century access control be. We speak primarily about the authorization aspect of access control. One approach is exemplified by XACML [19], an XML implementation of a limited access control language. A more ambitious approach is logic based, and that is where Distributed Knowledge Authorization Language (DKAL) belongs.

Access control is about knowledge and information, and these two concepts are central in DKAL. Pieces of information are called infons and are treated as elements of sort Info. Knowledge is formalized by means of a binary relation `knows` of type Principal $\times$ Info.

The DKAL story starts with a logic-based authorization language SecPAL that attracted our attention. We wanted to strengthen and improve SecPAL. In DKAL 1 [9], a security hole in SecPAL was plugged by making communication directed. The straightjacket of Constraint Datalog was loosened to Liberal Datalog where one can freely use functions in the heads and bodies of rules. These and other advances have been achieved within the same bounds of computational complexity.

In the meantime DKAL 1 has been successfully tried out at Microsoft which deepened our understanding of the issues involved and led to further improvements, notably to the following separation of concerns: static — and often small — core policies on the one side, and dynamic — and often complex — workflow on the other side [10]. It turns out that some important safety properties can be proved from the core policies alone, independently from the workflow.

And yet there was an important need that DKAL 1, or for that matter any other authorization language that deals with communication, could not meet. It is the need to communicate information conditioned on a proviso to be checked by the receiver rather than by the sender. For example, parents may will their money to their son provided he gets a college degree; the executors of the will would have to verify that the son has a degree before giving him the money. Here is a less grievous example. An author may stipulate that her book may be sold by a bookshop provided they have a positive rating from the Better Business Bureau.

The desire to meet that need was the major (but not the only) stimulus to revise DKAL. The result is DKAL 2 that we present here, a substantially more expressive and surprisingly simpler version of the language.

One of the novel features of DKAL 1 was an algebra of infons. Infons are partially ordered and equipped with the operation of addition. The intuition behind $x \leq y$ is that the information in x is contained in that of y. (Formally $x \leq y$ if x is derived from y by certain rules.) The intuition behind $x \wedge y$ is that the information in $x \wedge y$ is the union of that in x and y. In terms of the information order, $x \wedge y$ is the least upper bound of x and y. In order to convey information conditioned on a proviso, we introduced conditional infons $x \to y$. The intended meaning of $x \to y$ is y given x. In terms of the information order, $x \to y = \min\{z : x \wedge z \geq y\}$.

Eventually we switched from algebra to logic that treats $x \wedge y$ and $x \to y$ as conjunction and implication respectively. And something unexpected happened, a little miracle. The logic of infons happened to be a natural and conservative extension of disjunction-free intuitionistic logic. Trust became definable. For example, "p trusted on saying x" means $(p \texttt{ said } x) \to x$. The natural properties of trust, postulated as house rules of DKAL 1, are now *proved*. In fact, none of the house rules of DKAL 1 is now needed. Also, even though DKAL 2 is much more expressive than DKAL 1, the termination problem (for the derivation engine) became easier. Those who read the DKAL 1 paper [9] may remember that the termination proof was hard. In the case of DKAL 2, termination follows from general logic considerations; see §5 below.

The communication apparatus of DKAL 2 is also more powerful than that of DKAL 1. Following [10], we give principals means to filter incoming information. DKAL 2 is really a platform for authorization logic and communication.

DKAL 2 supersedes DKAL 1 and supports new important scenarios. There is, however, a caveat. We dropped the constructs `can act as` and `can speak as` that were supposed to be used to express roles. But we have never used them. Roles can be expressed more adequately by means of attributes which allows one to parameterize roles. In our experience, roles are always parameterized.

There is a price though for the extra expressivity of DKAL 2, at least in the terms of worst case complexity. Define the *ground multiple derivability problem* for a logic L to be the problem to compute, given ground hypotheses $x_1, \ldots, x_m$ and queries $y_1, \ldots, y_n$, which of the queries are derivable from the hypotheses. In the case $n = 1$, we speak about the *ground derivability problem*. The ground derivability problem for intuitionistic logic is polynomial space complete in the worst case [17]. For the same reasons, the ground derivability problem for infon logic is polynomial space complete in the worst case.

On the other hand, in our experience, the derivability problem has been consistently easy. Are there theoretical reasons for that? One reason is simple and obvious. While DKAL allows us to nest quotations arbitrarily

$p_1 \texttt{ said } p_2 \texttt{ said } \ldots p_k \texttt{ said } foo,$

in applications quotation depth is typically small. The analysis of typical derivations led us to a deeper (and orthogonal) reason. We came up with a fragment of infon logic that we call primal infon logic [11]. Primal infon logic looks simple, even simplistic. But it is quite expressive for our purposes. Let L_d be the fragment of primal infon logic with infons of quotation depth $\leq d$. In §9, we translate SecPAL into L_2. And yet, for any natural number d, the ground multiple derivability problem for L_d is decidable in linear time [11]. This gives rise to a linear time algorithm for the ground multiple derivability problem for SecPAL; see §9.

We attempted to make this paper as self-contained as possible but the familiarity with the concepts of knowledge and infons from DKAL 1 is desired. Here is the annotated context of the paper. The syntax of DKAL 2 is explained in §2. In §3, we explained how principals acquire knowledge. §4 is an introduction to and motivation for infon logic. To make room for genuinely access control

issues, we separated results of purely logical and algorithmic nature into a separate logic paper [11] but, to make this paper more self-contained, we recall some of those results here. In §5 we recall a sequent calculus, natural model theory and the soundness and completeness theorem for infon logic. The calculus has the *subformula property*: if a sequent $[\Gamma \vdash x]$ is provable then there is a derivation of the sequent that uses only the subformulas of the formulas in $\Gamma \cup \{x\}$. It is because of that property the derivation algorithm runs in polynomial space and in particular terminates. In §6, we define two versions of trust and prove various "house rules" of DKAL 1.

In §7, we illustrate DKAL on a scenario where a policy pointer is attached to an object. The policy must be respected as the object moves, in our case to a user Alice. Such scenarios are of increasing importance, in particular to organizations like TSCP [18], as authorization policies are expected to capture confidentiality, use, and copyright restrictions of documents traversing distant locations. Our example, despite its length, is terse relative to the real life scenarios. Because of space limitation it is substantially simplified. It concentrates on several features of DKAL2: using conditional infons and tagged variables, functions to communicate a proviso to be evaluated by the receiver; the use of casting of truth values into infons in order to express constraints; the distributed nature of DKAL and the interaction of logic and communication; and finally the use of filter assertions in guarding against information leakage.

§8 is devoted to primal infon logic. In §9 we translate SecPAL to a fragment of primal infon logic and construct a linear time algorithm for the ground multiple derivability problem for SecPAL. The final §10 is devoted to related literature.

2 Syntax

Basic syntax DKAL universe has elements of two types, Regular, and Synthetic. Both are divided into subtypes. The regular subtypes include the types Boolean (which consists of two elements, `true` and `false`), Principal, and possibly other subtypes depending on the application. The synthetic subtypes are Attribute, Speech, and Info. Elements of Info are called infons. There are four subtypes of Info: AttributeInfo, SpeechInfo, ConditionalInfo, and SumInfo.

We presume that each regular element is the value of a ground regular term. We often identify the term and the element it names.

Functions are either synthetic, meaning that they take synthetic values, or regular, meaning that they take regular values. The synthetic functions are free constructors; every synthetic element is uniquely constructed from regular elements by means of synthetic functions.

The following *built-in* synthetic functions are always present:

- Function $\wedge$ of type Info$\times$Info$\to$SumInfo.
- Function $\to$ of type Info$\times$Info$\to$ConditionalInfo.

- Functions `said` and `implied` of type Info→Speech.
- Function $\mathcal{I}$ of type (Regular×Attribute ∪ Principal×Speech) →Info.
- A nullary function `exists` of type Attribute.
- A nullary function `asInfon` of type Attribute.

There may be other synthetic functions, depending on the application. They take Attribute values. We refer to them as *attribute names*.

Infix notation is used for conjunction $x \wedge y$ and implication $x \rightarrow y$. Prefix notation is used for `said` and `implied`. The function $\mathcal{I}$ is usually omitted. For example, $\mathcal{I}(p, \texttt{said foo})$ is written simply as p `said foo`. We often omit parentheses in writing terms, so long as the intention is clear. Here is a summary of the syntax where x is an arbitrary term, u is a regular term and p is a principal term.

attrName `::= exists | asInfon`
`| can read [-] | can perform [-] on [-] | ...`

attribute `::=` *attrName*$(x_1, \ldots, x_n)$

speech `::= said` *infon* `| implied` *infon*

infon `::=` *attributeInfo* `|` *speechInfo* `|` *sumInfo* `|` *condInfo*

attributeInfo `::=` u *attribute*

speechInfo `::=` p *speech*

sumInfo `::=` *infon*$_1$ $\wedge$ *infon*$_2$

condInfo `::=` *infon*$_1$ $\rightarrow$ *infon*$_2$

The ensue relation DKAL has a relation $x \leq \Gamma$, read x ensues Γ, of type Info×Set(Info). The intuitive meaning is that x is less informative than Γ, or equally informative. Precise semantics, given in §5 approximates this intuition using deduction. Instead of $x \leq \Gamma$ we often write $\Gamma \vdash x$ and say that Γ entails x.

Verbatim tags Occurrences of regular variables and functions (including nullary functions, that is constants) in a term t can be tagged *verbatim*. We call the term t itself tagged it at least one verbatim tag occurs in t. If t is a tagged regular term $f(t_1, \ldots, t_j)$ we require that the exhibited occurrence of f is tagged. Thus a regular term $f(t_1, \ldots, t_j)$ is tagged if and only if the exhibited occurrence of f is tagged.

It follows that any synthetic or tagged regular term t is constructed from tag-free regular subterms and tagged variables by means of synthetic and tagged regular function. These tag-free regular subterms, the maximal tag-free regular subterms of t, are the *tag-free regular components* of t. If t itself is tag-free then the tag-free regular components of x are *regular components* of x. This last

definition is more meaningful when t is synthetic; if a regular term is tag-free than it is its only regular component.

Verbatim tags are used in communication assertions (see below) to indicate the variables and functions to be instantiated and evaluated by the receiver. Precise semantics of this will be given in §3. By default terms are tag-free.

Assertions There are three kinds of DKAL assertions: *knowledge*, *communication*, and *filter assertions*. Knowledge assertions have the form:

$A:\ x$

where A is a principal constant and x is a tag-free infon term with only regular variables. Communication assertions have the form

$A\ \texttt{to}\ q:\ [x \longleftarrow y] \Leftarrow z$

where A is a principal constant, q is a principal constant or variable, and x, y, z are infon terms with only regular variables such that x and y may be tagged but z is tag-free. y and z are optional. Filter assertions have the form

$B\ \texttt{from}\ p:\quad [x \longleftarrow y] \Leftarrow z$

where B is a principal constant, p is a principal constant or variable, and x, y, and z are tag-free infon terms. x and y may have synthetic as well as regular variables. y and z are optional.

3 Knowledge

Each principal A computes his (her, its) own knowledge. The knowledge assertions of A form an internal source of knowledge; this issue is addressed in §3.1. There may be also communications to A from other principals. Using his filter assertions A may extract knowledge from those communications; this issue is addressed in §3.2. Infon logic allows A to derive more knowledge; this issue is addressed in §3.3. And of course A may use his communication assertion to send communications to other principals; those communications may allow them to acquire knowledge.

Fix a state X of a principal A, and let $R_X(A)$ be the least set of regular terms such that the following conditions are satisfied.

- $R_X(A)$ contains constant A. For every assertion α (of any kind) of A and every infon term s in α, $R_X(A)$ contains the ground regular components of s.
- If, at state X, A knows an infon ($B\ \texttt{said}\ x$) or ($y \rightarrow B\ \texttt{implied}\ x$) as a direct result of a previous communication then $R_X(A)$ contains the ground regular components of the infon.
- If, at state X, A has a communication from B then — whether the communication will or will not be accepted — $R_X(A)$ contains B.

It will be convenient to identify ground terms to be evaluated at a fixed state with their values at the state. In particular we may see $R_X(A)$ as a set of regular elements. Furthermore, equalities $s = t$, of terms to be evaluated at a fixed state, will be by default interpreted as the equalities of the values of the terms.

3.1 Knowledge knowledge assertions

A knowledge assertion of A has the form $A : s$ where s is a tag-free infon term with regular variables only. The assertion gives rise to A's knowledge of various instances of the infon term s:

(KA) $$\frac{A : s}{A \texttt{ knows } s\,\theta}$$

Here θ is an arbitrary substitution subject to the following constraints.

(T1)
- $Dom(\theta)$ contains all variables of s,
- $Range(\theta) \subseteq R_X(A)$,
- $u\,\theta \in R_X(A)$ for every non-ground maximal regular component u of s.

3.2 Knowledge from communication

A communication assertion of a principal B has the form

$$B \texttt{ to } p : \ [t_1 \longleftarrow t_2] \Leftarrow t_3$$

where t_1, t_2, t_3 are infon terms with regular variables only and t_3 is tag-free. Principal B operates in a state Y of his. The communication assertion gives rise to communications from B to A by means of substitutions η such that $B \texttt{ knows } t_3\,\eta$ and $p\,\eta = A$. Also, this is required:

(T2)
- $Dom(\eta)$ contains all variables with untagged occurrences in $[t_1 \longleftarrow t_2] \Leftarrow t_3$,
- $Range(\eta) \subseteq R_Y(B)$,
- $u\,\eta \in R_Y(B)$ for every non-ground maximal regular component u of any t_i.

The resulting communication is

(C) $$B \texttt{ to } A : \ [u_1 \longleftarrow u_2]$$

where $u_1 = t_1\,\eta$ and $u_2 = t_2\,\eta$ but of course A does not in general know the substitution η or the original communication assertion of B.

The term u_2 is the *proviso* of (C). All variables in u_1, u_2 are regular and tagged. A may extract knowledge from communication (C) by means of a filter

assertions

(F) A `from` $q : [s_1 \longleftarrow s_2] \Leftarrow s_3$

of his and a substitution θ such that

(T3)
- $Dom(\theta)$ contains all variables in $[s_1 \longleftarrow s_2] \Leftarrow s_3$ and in u_1, u_2;
- θ treats the tagged and untagged versions of a variable as two distinct variables;
- $Range(\theta) \subseteq R_X(A)$,
- $u\,\theta \in R_X(A)$ for every non-ground maximal regular component u of any s_i;

and the following conditions are satisfied.

- If communication (C) does not have the optional proviso u_2 then the filter assertion (F) does not have the optional premise s_2.
- A `knows` $s_3\,\theta$, and $q\,\theta = B$.
- $s_1\,\theta = Untag(u_1\,\theta)$ and in the proviso-present case, $s_2\,\theta = Untag(u_2\,\theta)$.

And what is the resulting knowledge? The answer depends on the presence of a proviso. In the absence of proviso, it is

$$A \texttt{ knows } B \texttt{ said } s_1\,\theta.$$

In the presence of proviso, it is

$$A \texttt{ knows } (s_2\,\theta \rightarrow B \texttt{ implied } s_1\,\theta).$$

We summarize the two scenarios (ignoring conditions (T2) and (T3)) from the point of view of an outside observer, say an auditor:

`Proviso-free scenario:`

(Com1)
$$\frac{\begin{array}{l} B \texttt{ to } p : \ [t_1] \Leftarrow t_3, \\ B \texttt{ knows } t_3\,\eta, \quad p\,\eta = A, \\ A \texttt{ from } q : \ [s_1] \Leftarrow s_3, \\ A \texttt{ knows } s_3\,\theta, \quad q\,\theta = B, \\ s_1\,\theta = Untag(t_1\,\eta\,\theta) \end{array}}{A \texttt{ knows } B \texttt{ said } s_1\,\theta}$$

`Proviso-present scenario:`

(Com2)
$$\frac{\begin{array}{l} B \texttt{ to } p : \ [t_1 \longleftarrow t_2] \Leftarrow t_3, \\ B \texttt{ knows } t_3\,\eta, \quad p\,\eta = A, \\ A \texttt{ from } q : \ [s_1 \longleftarrow s_2] \Leftarrow s_3, \\ A \texttt{ knows } s_3\,\theta, \quad q\,\theta = B, \\ s_1\,\theta = Untag(t_1\,\eta\,\theta) \\ s_2\,\theta = Untag(t_2\,\eta\,\theta) \end{array}}{A \texttt{ knows } (s_2\,\theta \rightarrow B \texttt{ implied } s_1\,\theta)}$$

Remark 3.1. A public function, e.g. an addition of numbers, can be tagged verbatim in u_i because B may not know the values of all the arguments. It is possible also that a function is tagged verbatim because A and B have different interpretations of it. B may not know A's interpretation in which case he might be unable to apply the function even if he knows the values of all the parameters.

Remark 3.2. One may be tempted to unify (C) and (F). Take into account, however, that regular functions are not necessarily free constructors.

3.3 Derived knowledge

The knowledge that A gets from his knowledge assertions and the communications from the other principals gives rise to more knowledge.

$$\text{(Ensue)} \qquad \frac{A \texttt{ knows } \Gamma \quad \Gamma \vdash x}{A \texttt{ knows } x}$$

The derived knowledge is infinite but of course A does not compute it all. Typically he just checks whether some queries of interest follow from the knowledge obtained directly from knowledge assertions and communications.

Remark 3.3. As the receiver principal A gets a communication (C) from another principal B, A does not have to immediately apply all his filter assertions (F) and all possible substitutions θ to (C). He applies different filters and substitutions as the need arises. But what about the sender B? Does he have to use all his communication assertions with all possible substitutions η? No, not at all. Much depends on the workflow. One very common scenario is where the sender only replies to requests.

4 Infon logic: a Hilbert-type calculus

Intentionally $x \wedge y$ is the union of x and y, and $x \to y$ is y given x. We discovered that, as far as these two operations are concerned, the logic of infons is precisely intuitionistic logic, with sum playing the role of conjunction. Accordingly we fashion the logic of infons as an extension of intuitionistic logic.

The traditional Hilbert-type calculus for intuitionistic logic without disjunction and negation is given by the following five axioms and one inference rule called modus ponens [12].

`1a.` $x \to (y \to x)$

`1b.` $(x \to y) \to ((x \to (y \to z)) \to (x \to z))$

`2a.` $x \to (y \to (x \wedge y))$

`2b.` $(x \wedge y) \to x$ `2c.` $(x \wedge y) \to y$

`3.` $$\frac{x \quad (x \to y)}{y}$$

Think of the variables as infon variables. Uninformative infons play the role of true propostions. It is easy to see that axioms 1a–2c are uninformative and that the inference rule 3 preserves the property of being uninformative.

A set Γ of hypotheses *entails* an infon x if there is a chain $x_1, \ldots, x_n$ with $x_n = x$ where every member is an axiom, a hypothesis or the result of an inference rule application to earlier members. It is this definition of entailment that characterizes Hilbert-type calculi.

Our Hilbert-type infon calculus is the extension of the intuitionistic calculus above by means of the following axioms 4a–7.

4a. $(p \texttt{ said } (x \to y)) \wedge (p \texttt{ said } x) \to (p \texttt{ said } y)$

4b. $(p \texttt{ implied } (x \to y)) \wedge (p \texttt{ implied } x) \to (p \texttt{ implied } y)$

5. $(p \texttt{ said } x) \to (p \texttt{ implied } x)$

6. $\texttt{true asInfon}$

7. $p_1 \texttt{ said } \ldots\ p_k \texttt{ said } x$ where $p_1, \ldots, p_k$ is any finite sequence of principles and x is any of the axiom 1a–6

To understand the intuition behind axioms 4a and 4b, think of a principal q listening to principal p. Things said by p are closed under deduction, and so are things implied by p. The same intuition lies behind axiom 7. By axiom 5, `said` is stronger than `implied`. We'll speak more about this later. Axiom 5 explains why axiom 7 is restricted to `said`. If you replace some of the k `said`'s with `implied`'s, the result is derivable. Concerning axiom 6, we think of the operation $\texttt{true} \mapsto \texttt{true asInfon}$ as a casting operation, converting Boolean `true` into the uninformative infon `true asInfon`.

Note that the built-in function `exists` is not constrained by axioms. It allows one to mention a regular element and thus make it count; no axioms are needed for that. Let $q \texttt{ said } \Delta = \{q \texttt{ said } x : \ x \in \Delta\}$.

Lemma 4.1. *If $\Delta \vdash y$ then q `said` $\Delta \vdash q$ `said` y.*

Proof. If $\Delta = \emptyset$, use axiom 7. Suppose that $\Delta \neq \emptyset$, and let $x_1, \ldots, x_n$ be the given derivation of y from Δ. Then the chain $q \texttt{ said } x_1, \ldots, q \texttt{ said } x_n$ can be filled in to a derivation of $q \texttt{ said } y$ from $q \texttt{ said } \Delta$. Indeed, it suffices to show this: if x_k is obtained, by an application of modus ponens, from previous members x_i and $x_j = x_i \to x_k$, then

$$q \texttt{ said } x_i,\ q \texttt{ said } (x_i \to x_k) \vdash q \texttt{ said } x_k.$$

But this easily provable by means in particular of axiom 4a. □

Let `told`, with or without indices, range over {`said`, `implied`}. If $\Delta = \{z_1, \ldots, z_m\}$, let $q \texttt{ told } \Delta = \{q \texttt{ told}_i \ z_i : 1 \leq i \leq m\}$.

Lemma 4.2. *If $\Delta \vdash y$ then q `told` $\Delta \vdash q$ `implied` y.*

Proof. Assume $\Delta \vdash y$. Use axiom 5 to derive Δ `implied` y from q `told` Δ. The rest of the proof is similar to the proof of the previous lemma, with `implied` playing the role of `said`. □

5 Infon logic: a sequent calculus

It is advantageous to treat the relation $\Gamma \vdash x$ as primitive and define it by means of axioms and rules of inference. That approach is common in the study of intuitionistic logic. Pairs (Γ, x) are called sequents, and the calculi of sequents are called sequent calculi. In this section, we present a sequent calculus S equivalent to the Hilbert-type calculus of the previous section.

In Section 2, we mentioned the ensue relation $x \leq \Gamma$. Relation $\Gamma \vdash x$ is the same relation written in a different way: $x \leq \Gamma$ if and only if $\Gamma \vdash x$.

5.1 Axioms and rules of inference

Our sequent calculus S is essentially an extension of the intuitionistic propositional system NJp [16, §2.2].

Axioms

`(True)` $\quad \vdash$ `true asInfon`

`(x2x)` $\quad x \vdash x$

Inference rules

`(Premise Inflation)` $\dfrac{\Gamma \vdash y}{\Gamma, x \vdash y}$

`(`$\wedge$`E)` $\dfrac{\Gamma \vdash x \wedge y}{\Gamma \vdash x}$ $\qquad$ $\dfrac{\Gamma \vdash x \wedge y}{\Gamma \vdash y}$

`(`$\wedge$`I)` $\dfrac{\Gamma \vdash x \quad \Gamma \vdash y}{\Gamma \vdash x \wedge y}$

`(`$\rightarrow$`E)` $\dfrac{\Gamma \vdash x \quad \Gamma \vdash x \rightarrow y}{\Gamma \vdash y}$

`(`$\rightarrow$`I)` $\dfrac{\Gamma, x \vdash y}{\Gamma \vdash x \rightarrow y}$

`(S)` $\dfrac{\Delta \vdash y}{q \texttt{ said } \Delta \vdash q \texttt{ said } y}$

`(P)` $\dfrac{\Delta \vdash y}{q \texttt{ told } \Delta \vdash q \texttt{ implied } y}$

From the Liberal Datalog point of view [9, 3], the axioms and inference rules form a program that computes a binary relation $\vdash$ of type `Set(Info)` $\times$ `Info`.

5.2 Model theory

A Kripke structure for infon logic, given by the sequent calculus, is a non-empty quasi-order $(W, \leq)$ of worlds with two binary relations P_q and S_q for every principal q such that the following requirements are satisfied.

K1. $P_q \subseteq S_q$.

K2. If $u \leq w$ and wP_qv then uP_qv, and the same for S_q.

By induction, any formula x is assigned a cone (that is an upward closed set) $C(x)$ of worlds. If $u \in C(x)$, we say that x holds in u or that u models x, and we write $u \models x$. If x is primitive then $C(x)$ is an arbitrary cone. Further:

K3. $C(x \wedge y) = C(x) \cap C(y)$.

K4. $C(x \to y) = \{u : C(x) \cap \{v : v \geq u\} \subseteq C(y)\}$.

K5. $C(q \texttt{ implied } x) = \{u : \ \{v : \ uP_qv\} \subseteq C(x) \ \}$.

K6. $C(q \texttt{ said } x) = \{u : \ \{v : \ uS_qv\} \subseteq C(x) \ \}$.

K7. $C(\texttt{True}) = W$.

Clauses K3, K4 and K7 are standard in the Kripke semantics for intuitionistic logic and go back to [13].

It is easy to check, by induction on formula z, that every $C(z)$ is indeed a cone. We consider here only the case when $z = (q \texttt{ implied } x)$. It suffices to show that $\{v : wP_qv\} \subseteq \{v : uP_qv\}$ given that $u \leq w$. But this follows from K2.

Further let $C(\Gamma) = \bigcap_{x \in \Gamma} C(x)$; in particular $C(\emptyset) = W$. A Kripke structure K models a sequent $(\Gamma \vdash x)$ if $C(\Gamma) \subseteq C(x)$ in K. A sequent s is *valid* if every Kripke structure models s.

Theorem 5.1 ([11])**.** *The following claims are equivalent for any sequent s.*

1. *s is provable in the sequent calculus.*
2. *s is valid.*
3. *Every finite Kripke structure models s.*
4. *There is a proof of s in the sequent calculus that uses only subformulas of s.*

5.3 The two calculi are equivalent

Theorem 5.2. *Γ entails x in the Hilbert-type calculus of Section 4 if and only if sequent $(\Gamma \vdash x)$ is derivable in the sequent calculus.*

Proof. Let H be the Hilbert-type calculus. First we establish the only-if direction. In virtue of Theorem 5.1, it suffices to show that every model M of Γ is a model for every x entailed by Γ in H. It is easy to see that the axioms of H are valid and thus hold in M. It remains to notice that the set of infons true in M is closed under modus ponens.

Next we establish the if direction. It suffices to show that every inference rule of the sequent calculus can be proved as a derived rule in H. This is obvious for the premise inflation rule and for rules ($\wedge$E), ($\wedge$I), ($\to$E). The rule ($\to$I) is the deduction theorem [12]. The rules (S) and (P) are proved in Lemmas 4.1 and 4.2. □

6 Trust and delegation

DKAL2 has two trust functions: `tdonS` and `tdonI`. Both are of type Principal $\times$ Info. Read infons p `tdonS` x and p `tdonI` x as p is trusted on saying x and p is trusted on implying x respectively. But the two trust functions are not primitive notions of DKAL2; they are defined:

p `tdonS` x abbreviates `(`p `said` x`)` $\to x$,

p `tdonI` x abbreviates `(`p `implied` x`)` $\to x$.

Proposition 6.1.

p `tdonS` x, p `tdonS` $y \vdash p$ `tdonS` $(x \wedge y)$.

p `tdonI` x, p `tdonI` $y \vdash p$ `tdonI` $(x \wedge y)$.

Proof. We prove the case of saying; the case of implying is similar. By rule ($\to$I), it suffices to derive $x \wedge y$ from the set Γ of hypotheses p `tdonS` x, p `tdonS` y and p `said` $(x \wedge y)$. By ($\wedge$I), it suffices to show that $\Gamma \vdash x$ and $\Gamma \vdash y$. We show that $\Gamma \vdash x$; the case of y is similar.

By ($\wedge$E), $x \wedge y \vdash x$. By (S), p `said` $(x \wedge y) \vdash p$ `said` x. Hence $\Gamma \vdash p$ `said` x. Use this sequent as the minor premise of ($\to$E) to obtain $\Gamma \vdash x$. □

Let $(p$ `seconds` $x)$ abbreviate infon $(x \to (p$ `implied` $x))$.

Proposition 6.2 (Delegation of trust on implying)**.**

p `tdonI` x, p `seconds` q `implied` $x \ \vdash \ p$ `tdonI` q `tdonI` x.

Proof. Expand the premises and the conclusion. The latter becomes

$$(p \text{ `implied` } ((q \text{ `implied` } x) \to x)) \to ((q \text{ `implied` } x) \to x).$$

By ($\to$I), it suffices to derive x from the original premises
`a.` $(p$ `implied` $x) \to x$,
`b.` $(q$ `implied` $x) \to p$ `implied` q `implied` x

and the successive premises of the expanded conclusion

c. p `implied` $((q$ `implied` $x) \to x)$,
d. q `implied` x.

By b and d, get p `implied` q `implied` x. From this and c, get p `implied` x. From this and a, get x. □

Proposition 6.3 (Del$^-$)**.**
p `tdonS` p `tdonS` $x \vdash p$ `tdonS` x,
p `tdonI` p `tdonI` $x \vdash p$ `tdonI` x.

Proof. We prove the first claim; the other is similar. It suffices to derive x from the given premise, whose expanded form is

a. $(p$ `said` $((p$ `said` $x) \to x)) \to$
$((p$ `said` $x) \to x)$,

and the premise

b. p `said` x.

of the (expanded) conclusion.

Since x, p `said` $x \vdash x$, use ($\to$I) to obtain $x \vdash (p$ `said` $x) \to x$. By (S), p `said` $x \vdash p$ `said` $((p$ `said` $x) \to x)$, that is b. entails the premise of a. Use this and b. to get the premise of a. Use a. and the premise of a. to get the conclusion of a. Use b. and the conclusion of a. to obtain x. □

7 Example

Dramatis personae:

`Alice`, a buyer
`Publishers`, an organization of publishers
`Best`, a publisher
`Bob`, a hacker
`Bureau`, a service rating organization
`Chux` (an allusion to Chuck's), a seller
`Integral`, a certification company

Public substrate functions:

object `isGovernedBy` policy_name `of` authority
returns a truth value
$\text{time}_1 < \text{time}_2$
returns a truth value
`curTime()`
returns time and date

Attributes:

`mayDo` action `on` object
`is a licensed seller`
`accedes to purchase` object
(accepting the associated rights and obligations)
`has good standing`

Now we are ready to proceed to the story. Alice wishes to play a rare Song and is willing to pay for that, but she wants to make sure that the deal is proper and in accordance with the law. She has a knowledge assertion:

```
Alice:   s isGovernedBy pol of auth →
       auth tdonI Alice mayDo a on s
```

We abbreviate the awkward "mayDo PLAY on s" to "may play s". Alice learns that

```
Song is governed by Policy A of Publishers
```

and so she knows

(A1) `Publishers tdonI Alice may play Song`

which, by the definition of `tdonI`, is equivalent to

```
(Publishers implied Alice may play Song) →
(Alice may play Song).
```

Alice gets a communication from Publishers:

```
Publishers to Alice:  [Alice may play Song ←
   c^v is a licensed seller ∧ c^v said Alice may play Song].
```

Here c^v is a verbatim variable to be substituted by the receiver. Alice uses seller Chux and accepts any proviso-free communication from Publishers. The latter means that she has a filter assertion

```
Alice from Publishers:  [x]
```

where x is an infon variable. Accordingly she learns

(A2)
```
Chux is a licensed seller ∧
Chux said Alice may play Song →
Publishers implied Alice may play Song.
```

In full infon logic, implications (A1) and (A2) give

```
Chux is a licensed seller ∧ Chux said Alice may play Song →
Alice may play Song
```

which is equivalent to

```
Chux is a licensed seller → Chux tdonS Alice may play Song
```

but let us stick to primal logic. Alice has a knowledge assertion

(A3) `Bureau tdonI c is a licensed seller.`

And indeed, licenses are regulated by Bureau that maintains a database of licensed sellers and license expiration dates using substrate functions

```
isLicensed :  sellers to Bool
LicExp :  sellers to Time/Date.
```

They have a communication assertion

```
Bureau to Alice:
   [c is a licensed seller ⟵ CurTime^v() < LicExp(c)] ⇐
   c isLicensed.
```

Notice the verbatim nullary function CurTime^v() to be evaluated by receivers. Notice also that constraint `CurTime`^{v}`() < LicExp(c)` has been implicitly converted to infon `[CurTime`^{v}`() < LicExp(c)] asInfon`. Chux happens to be licensed with expiration date of 1/1/2012. Accordingly Alice gets communication

```
Bureau to Alice:
   CurTime^v() < 1/1/2012 →
   Bureau implied Chux is a licensed seller.
```

The premise is satisfied as it evaluates to `(true asInfon)` which is an axiom. Alice does not filter Bureau's communications and and so she learns

```
Bureau implied Chux is a licensed seller.
```

Taking into account (A3) she learns

(A4) `Chux is a licensed seller.`

Chux uses a communication assertion

```
     Chux to p:  [p may play s] ⇐
(C0)    p implied p accedes to purchase s ∧
        Integral said p has good standing.
```

Being in constant contact with Integral, Chux easily checks the second condition. Alice happens to have a good standing according to Integral. But Chux requires that clients agree to purchase items before they can use them. Alice communicates:

```
Alice to Chux:
   [Alice accedes to purchase Song].
```

This is filtered through

(C1) `Chux from p:  p accedes to purchase s`

giving rise to Chux knowing

```
Alice said Alice accedes to purchase Song.
```

Chux issues communication

```
Chux to Alice:  [Alice may play Song],
```

and so Alice learns

(A5) `Chux said (Alice may play Song).`

Now, (A2), (A4) and (A5) give

```
Publishers implied (Alice may play Song).
```

Taking (A1) into account, Alice gets

```
Alice may play Song.
```

A cautionary note on divulging information

Concerning (C1), Chux can be more general. They can even assert

(C2) `Chux from p:  [x]`

which means only that they are willing to learn `p said x` regardless of what x is. This implies no trust in p on saying x and thus does not lead, all by itself, to the knowledge of x. But a blanket assertion:

(C3) `Chux from p:  [x` $\longleftarrow$ `y]`

amounts to divulging information (though they may have a narrow version of (C3) where p is bound e.g. to the departments of Chux). To illustrate the problem with (C3), suppose that Chux knows already that Integral said that Bob has good standing. If Bob wishes to find out whether Alice also has good standing with Integral, he can try this:

```
      Bob to Chux:
(B1)     [Bob accedes to purchase Song ⟵
         Integral said Alice has good standing].
```

By checking whether Chux allows him to play Song, Bob could indirectly check whether Chux knows that Integral said Alice has good standing. Indeed suppose that Chux knows the proviso of (B1). Then, by (B1),

```
Chux knows Bob implied Bob accedes to purchase Song
```

and, by (C0), Bob gets a communication

```
Bob may play Song.
```

The communication is not issued by Chux if they do not know the proviso of (B1).

8 Primal infon logic

Primal infon logic [11] is a fragment of infon logic that is useful in practice. In contrast the infon logic of §4 and 5 will be called full infon logic. To make this paper more self-contained, we recall some some basic facts about primal logic. To avoid confusion, one may want to use a different entailment sign, e.g. $\vdash^p$, for primal infon logic. But we will be dealing only with primal infon logic in the rest of the paper and so we will use the usual entailment sign $\vdash$.

8.1 Axioms and rules of inference

We recall the Hilbert-type calculus for primal infon logic. (There is also an equivalent sequent calculus in the logic paper [11]). Let `pref` with or without a subscript range over strings of the form

$$q_1\ \texttt{told}_1\ q_2\ \texttt{told}_2\ \ldots q_k\ \texttt{told}_k$$

where `told` ranges over $\{\texttt{said}, \texttt{implied}\}$ and k may be zero. We write $\texttt{pref}_1 \leq \texttt{pref}_2$ if $\texttt{pref}_1$ is the result of replacing some (possibly none) occurrences of `said` in $\texttt{pref}_2$ with `implied`.

Axioms

$$\texttt{pref True}$$

Inference rules

(Pref Deflation) $\dfrac{\texttt{pref}_2\, x}{\texttt{pref}_1\, x}$ where $\texttt{pref}_1 \leq \texttt{pref}_2$

(Pref∧E) $\dfrac{\texttt{pref}\,(x \wedge y)}{\texttt{pref}\, x}$ $\quad$ $\dfrac{\texttt{pref}\,(x \wedge y)}{\texttt{pref}\, y}$

(Pref∧I) $\dfrac{\texttt{pref}\, x \qquad \texttt{pref}\, y}{\texttt{pref}\,(x \wedge y)}$

(Pref→E) $\dfrac{\texttt{pref}\, x \qquad \texttt{pref}\,(x \to y)}{\texttt{pref}\, y}$

(Pref→I) $\dfrac{\texttt{pref}\, y}{\texttt{pref}\,(x \to y)}$

8.2 Model theory

Kripke structures for primal infon logic are defined the same way as those for full infon logic except that the requirement K4 is replaced with a weaker — and nondeterministic — requirement

K4P. $C(x \to y)$ is an arbitrary cone such that
$C(y) \subseteq C(x \to y) \subseteq$
$\{u : \ C(x) \cap \{v : v \geq u\} \subseteq C(y)\}$.

Theorem 8.1 (Soundness and completeness; [11])**.** *The following claims are equivalent.*

1. $\Gamma \vdash \phi$ *in primal infon logic.*
2. *For any Kripke structure* K *for primal infon logic, if* Γ *holds in* K *then* ϕ *does.*
3. *For any finite Kripke structure* K *for primal infon logic, if* Γ *holds in* K *then* ϕ *does.*

8.3 Linear-time algorithms

The *ground multiple derivability problem* GMD(L) for a logic L is to compute, given ground hypotheses $x_1, \ldots, x_m$ and queries $y_1, \ldots, y_n$, which of the queries are derivable from the hypotheses.

We define the *quotation depth* of infons and set of infons. Note the unusual clause CD4.

CD1. $\mathrm{QD}(x) = 0$ if x is primitive.
CD2. $\mathrm{QD}(p \texttt{ told } x) = 1 \wedge \mathrm{QD}(x)$.
CD3. $\mathrm{QD}(x \wedge y) = \max\{\mathrm{QD}(x), \mathrm{QD}(y)\}$.
CD4. $\mathrm{QD}(x \to y) = \mathrm{QD}(y)$
CD5. $\mathrm{QD}(\Gamma) = \max\{\mathrm{QD}(x) : x \in \Gamma\}$.

Theorem 8.2. *Let* L_d *be the fragment of primal infon logic with infons of quotation depth* $\leq d$*. Every* GMD(L_d) *is solvable in linear time.*

9 SecPAL and primal infon logic

We consider SecPAL without the construct `can act as`. We could have easily incorporated the construct into DKAL 2 (and indeed it was incorporated into DKAL 1) which would allow us to deal with full SecPAL. But, as we explained in §1, we abandoned the construct because attributes with parameters are a finer tool to deal with roles. In the rest of this section, SecPAL means SecPAL without the `can act as` construct.

9.1 Embedding theorem

In the DKAL 1 paper, we translated SecPAL into a coarser version of DKAL 1 [9, §11] where principals broadcast whatever they know. In retrospect SecPAL is really about logic. Recall the bounded-quotation-depth fragments L_d of primal

infon logic introduced in Theorem 8.2. We translate SecPAL into a courser version L_2^0 of L_2 where every principal p seconds whatever is said or implied by any other principal q. More exactly, L_2^0 is an extension of L_2 with additional axioms

- `p seconds q said x,` that is
  `q said x` $\rightarrow$ `p implied q said x`
- `p seconds q implied x,` that is
  `q implied x` $\rightarrow$ `p implied q implied x`

Note that neither `said` nor `implied` can appear in x.

Any constraint domain of SecPAL is turned into a DKAL substrate in the obvious way, as in [9], except that now we treat relations as Boolean-valued functions and we presume that the substrate has the usual Boolean connectives, so that — over the substrate — SecPAL constraints are Boolean terms.

Translation

(Variable, constant) τ`(e) = e`
(Constraint) τ`(con) = con asInfon`
(Predicate) τ`(pred) = pred`
(Fact) τ`(e pred e`$_1$`,...,e`$_n$`) =`
`  `$\mathcal{I}$`(e, pred(e`$_1$`,...,e`$_n$`)) =`
`  e pred(e`$_1$`,...,e`$_n$`)`
τ`(e can say`$_0$` fact) =`
`  e tdonS` τ`(fact) =`
`  (e said` τ`(fact))`$\rightarrow$ τ`(fact)`
τ`(e can say`$_\infty$` fact) =`
`  e tdonI` τ`(fact) =`
`  (e implied` τ`(fact))`$\rightarrow$ τ`(fact)`
(Assertion)
τ`(A says fact if fact`$_1$`,...,fact`$_n$`,con) =`
`A said ((`τ`(fact`$_1$`)` $\wedge$ `...` $\wedge$ τ`(fact`$_n$`)` $\wedge$ τ`(con))`
`  `$\rightarrow$ τ`(fact))`

Thus SecPAL variables and constants become DKAL variables and constants respectively. Constraints become infon terms. Predicates become attribute names. Flat facts become attribute infon terms, and can-say facts become conditional infon terms. Finally SecPAL assertions become speech infon terms.

Theorem 9.1 (Embedding Theorem)**.**

1. *If* $AC, 0 \vdash A$ `says` f *in SecPAL, then*
   $\Gamma \vdash A$ `said` $\tau(f)$ *in* L_2^0.
2. *If* $AC, \infty \vdash A$ `says` f *in SecPAL, then*
   $\Gamma \vdash A$ `implied` $\tau(f)$ *in* L_2^0.

Here AC is a safe SecPAL assertion context, A is a principal constant, f is a ground fact expression, and Γ is the set of instantiations of speech terms in $\tau(AC)$ with constants that occur in AC or in A `says` f.

Proof. The proof is a routine induction on the given SecPAL derivation. Since AC is safe, we may — according to [4] — consider a derivation of A `says` f from ground instantiations of AC assertions with constants in AC or in A `says` f. We describe how to turn that derivation into a derivation of A `said` $\tau(f)$ from Γ in L_2^0.

SecPAL rule (cond) is simulated by the rule (Pref→E) of primal infon logic. SecPAL rule (can say) has two versions depending on whether D is 0 or ∞. We consider here only the case $D = 0$; the other case is similar. Suppose $AC, \infty \vdash A$ `says` B `can say`$_0$ f, and $AC, 0 \vdash B$ `says` f. By the induction hypothesis, we have

(a) $\Gamma \vdash A$ `implied` B `tdonS` $\tau(f)$,
(b) $\Gamma \vdash B$ `said` $\tau(f)$.

Claim (b) and the first additional axiom of L_2^0 give

(c) $\Gamma \vdash A$ `implied` B `said` $\tau(f)$.

By rule (Pref→E), the desired A `implied` $\tau(f)$ follows from (a) and (c). □

As in the case of the translation of SecPAL to DKAL 1, the converse of the embedding theorem fails. The counterexample to this effect, given in [9, §11], also applies in this case. Thus L_2^0 and therefore primal infon logic produce more true consequences than SecPAL.

9.2 Linear time algorithm

In §8.3 we defined the ground multiple derivability problem GMD(L) for an arbitrary logic L and we defined bounded-quotation-depth fragments L_d of primal infon logic.

Theorem 9.2. GMD(SecPAL) *is solvable in linear time.*

We can use Theorems 8.2 and 9.1 to solve GMD(SecPAL). The approach has a drawback, however. According to the translation table

τ(e `can say`$_0$ `fact`) = e `tdonS fact` =
  (e `said` τ(`fact`))→ τ(`fact`),

τ(e `can say`$_\infty$ `fact`) = e `tdonI fact`
  (e `implied` τ(`fact`))→ τ(`fact`).

Notice that the implication mentions `fact` twice. As a result the given hypotheses may be lengthened in the translation. It would be better not to expand abbreviations `tdonS`, `tdonI` and treat them as primitives. We will do that.

Proof sketch The fragment L_1 of primal infon logic allows one to quote (that is to use `said` and `implied`) but does not allow one to nest quotations. We proved that GMD(L_1) is solvable in linear time [11]. Here we prove Theorems 9.2 by modifying that proof, called the original proof below. We have to assume the familiarity with the original proof.

The (cond) rule of SecPAL is taken care of automatically by logic L_1. It is the rule (can say) than needs special care. The rule exists in two forms that can be expressed in our terms as follows:

(S1)
$$\texttt{(A implied B tdonS x)} \wedge \texttt{(B said x)} \rightarrow \texttt{A implied x.}$$
$$\texttt{(A implied B tdonI x)} \wedge \texttt{(B implied x)} \rightarrow \texttt{A implied x.}$$

We treat `tdonS` and `tdonI` as synthetic functions subject to conditions (S1). Expressions p `tdonS` x and p `tdonI` x are not abbreviations. Let, as in §5, `told` range over {`said`, `implied`}, and let p `tdonT` x mean that p is trusted on telling x. In other words, `tdonT` is `tdonS` if `told` = `said`, and `tdonT` is `tdonI` if `told` = `implied`.

Parsing There is a problem that needs to be taken care at this stage. It is possible that eventually we will derive the left part (A `implied` B `tdonT` x) $\wedge$ (B `told` x) of one of (S1) rules but A `implied` x is not present in the input. We remedy the problem by means of one depth-first traversal.

For every node u with label A `implied` do the following. For every node v with label B `tdonT` under u, attach an extra parent with label A `implied` to the child of v. The new node has no ancestors.

Cai-Paige algorithm As in the original proof, the Cai-Paige algorithm computes homonymy pointers $H(u)$. Adjust the algorithm to produce additional pointers as follows. Call homonymy originals u, v *mates* if their locutions form a pair B `tdonT` x and B `told` x. The mates are on the same height level. For every pair of mates, put pointers from one mate to the other. If only one of the two exists, the new pointer is null.

Table The table acquires another static field. If the label of u is B `tdonT` and the parent of u has label A `implied` then insert the parent into the new field of $H(u)$.

Processing Enrich the processing of pending formulas $L(u)$ taking into account rules (S1).

If $L(u)$ is B `told` x and the new pointer at u is not null, retrieve the mate v of u with locution B `tdonT` x. Go through the nodes w in new field of v such that, according to the status of w, the locution $L(w)$ has been proved. By construction $L(w)$ has the form A `implied` B `tdonT` x, and the grandchild of w has an extra parent w' with locution A `implied` x. If the status of $H(w')$ is raw, set the status of $H(w')$ to pending.

Suppose that $L(u)$ is A `implied` B `tdonT` x and let u_0 be the child of u. Then $L(u_0) = B$ `tdonT` x and so the child of u_0 has an extra parent u' with locution A `implied` x. If the new pointer at u_0 is not null, retrieve the mate v of u_0 with locution B `told` x. If the status of v shows that $L(v)$ has been proved but the status of $H(u')$ is raw, set the status field of $H(u')$ to pending. That completes the description of the algorithm. □

10 Related work

It was the Speaks-For calculus [1] that pioneered the logic based approach to access control, introducing among other things the `says` modality that has been used ever since. Our more recent genealogy consists primarily of Datalog based languages Binder [8], Delegation Logic [14], and especially SecPAL [4]. The DKAL 1 paper [9] summarizes the influence of these languages on our work. An additional language, SeNDlog [2] was developed independently from and roughly at the same time as DKAL 1. Like DKAL, it deals with both logic and communication, one of very few languages to address both issues. Logic in SeNDlog is handled directly by use of Datalog, and communication is handled by means of import and export predicates. The infon logic of DKAL 2 goes much beyond Datalog because of the use of functions and because of conditionals. Communication is also much enhanced in DKAL 2 by means of verbatim tags, by enabling the communication of conditional infons, and by the use of filter assertions to prevent information leakage that may result from communication of conditional infons.

Becker and Nanz addressed recently [5] an important issue of abduction: in the case of access denial, which authorization facts or credentials were missing that would have led to an access grant? We intend to extend the existing implementation of DKAL with abduction heuristics.

Work in the Trust Management framework [15] addresses policy analysis, including reachability and invariance checking. A similar analysis was done in the DKAL application paper [10]. The richness of DKAL makes these issues more challenging. The application paper [10] also achieved the true distributivity of DKAL. In SecPAL (the language rather than the implementation) principals are distributed but their sayings are collected and processed together. In DKAL 1 principals compute their own knowledge, yet some vestiges of a centralized approach remain, e.g. the global state. In [10] different principals live in different worlds exchanging information by means of communication and filtering assertions.

References

[1] Martín Abadi, Michael Burrows, Butler Lampson, and Gordon Plotkin, "A Calculus for Access Control in Distributed Systems," ACM Transactions on Programming Languages and Systems, 15:4, 706–734, 1993.

[2] Martín Abadi and Boon Thau Loo, "Towards a Declarative Language and System for Secure Networking", in International Workshop on Networking Meets Databases (NetDB '07), 2007.

[3] Andreas Blass and Yuri Gurevich, "Two Forms of One Useful Logic: Existential Fixed Point Logic and Liberal Datalog," *Bulletin of the European Association for Theoretical Computer Science* 95 (June 2008), 164–182.

[4] Moritz Y. Becker, Cédric Fournet and Andrew D. Gordon, "SecPAL: Design and Semantics of a Decetralized Authorization Language", 20th IEEE Computer Security Foundations Symposium (CSF), 3–15, 2007.

[5] Moritz Y. Becker and Sebastian Nanz, "The Role of Abduction in Declarative Authorization Policies," in 10th International Symposium on Practical Aspects of Declarative Languages (PADL), 2008.

[6] Jiazhen Cai and Robert Paige, "Using multiset discrimination to solve language processing problems without hashing," *Theoretical Computer Science* 145:(1-2), 189–228, July 1995.

[7] Thomas H. Cormen, Charles E. Leiserson and Ronald L. Rivest, *Algorithms,* MIT Press, 1990.

[8] John DeTreville, "Binder, a Logic-Based Security Language", in IEEE Symposium on Security and Privacy, 105-113, 2002.

[9] Yuri Gurevich and Itay Neeman, "DKAL: Distributed-Knowledge Authorization Language," 21st IEEE Computer Security Foundations Symposium (CSF 2008), 149–162.

[10] Yuri Gurevich and Arnab Roy, "Operational Semantics for DKAL: Application and Analysis," Microsoft Research Tech Report MSR-TR-2008-184, December 2008.

[11] Yuri Gurevich and Itay Neeman, "The Infon Logic," Microsoft Research Tech Report MSR-TR-2009-10, January 2009.

[12] Stephen Cole Kleene, "Introduction to Metamathematics" D. Van Nostrand Company, inc. 1952.

[13] Saul Kripke, "Semantical Analysis of Intuitionistic Logic I," in *Formal Systems and Recursive Functions*, eds. J. W. Addison, L. Henkin and A. Tarski, North-Holland 1965, 92–130.

[14] Ninghui Li, Benjamin N. Grosof and Joan Feigenbaum, "Delegation Logic: A Logic-Based Approach to Distributed Authorization", ACM Trans. on Information and System Security (TISSEC) 6:1 (February 2003), 128–171.

[15] Ninghui Li, John C. Mitchell, and William H. Winsborough "Beyond Proof-of-Compliance: Safety and Availability Analysis in Trust Management", in Proceedings of 2003 IEEE Symposium on Security and Privacy, 123–139, May 2003.

[16] Grigori Mints, *A Short Introduction to Intuitionistic Logic,* Kluwer Academic / Plenum Publishers 2000.

[17] Richard Statman, "Intuitionistic Propositional Logic is Polynomial-Space Complete," *Theoretical Computer Science* 9:1 (July 1979), 67–72.

[18] TSCP, `http://tscp.org/`, viewed Feb. 06, 2009.

[19] XACML, Extensible Access Control Markup Language, `http://xml.coverpages.org/xacml.html`, viewed Feb. 02, 2009.