AN INTRODUCTION TO DKAL

ANDREAS BLASS, GUIDO DE CASO, AND YURI GUREVICH

Tech Report MSR-TR-2012-108, October 2012
Microsoft Research Redmond

Abstract. We describe the current version of the Distributed Knowledge Authorization Language (DKAL) and some proposed extensions. The changes, in comparison with previous versions on DKAL, include an extension of primal infon logic to include a weak form of disjunction, the explicit binding of variables in infons and in rules, the possibility of updating a principal's policy, a more detailed treatment of a principal's datasources, and the introduction of DKAL communities.

The purpose of this report is to describe the present version of the Distributed Knowledge Authorization Language (DKAL), along with some proposed extensions. Previous versions were described in [10, 13, 11, 5]. New implementation-related developments are routinely reported at [8].

DKAL is a language for writing the policies of a distributed system's interacting agents, usually called *principals* in this context. Each DKAL principal performs its own computations and deductions; the system has no central logic engine but merely provides an infrastructure supporting reliable communication between principals. In DKAL, each principal works with a *substrate* containing various datasources. In addition, each principal works with knowledge that can be modified as a result of communications from other principals or as a result of deduction. We call this body of knowledge the principal's *knowledge base*. A principal has a *policy* telling it when and how to send messages, to update its substrate, and to accept or reject (or otherwise deal with) incoming messages. A policy may even provide, under certain circumstances, for updates to the policy itself. The knowledge base and policy together are sometimes called the principal's *infostrate*.

The first author is partially supported by NSF grant DMS-0653696 and a grant from Microsoft Research.

1. Principals

The principals in an interactive system, as described by DKAL, can be computers, computer systems, people, or organizations. In practice, principals other than computers will usually use computers for their actions, so we often pretend that the principals themselves are computers, at least by writing "it" rather than "he" or "she". We list here the capabilities of principals; we explain and discuss these matters in more detail afterward.

Principals can

(1) read information from certain datasources,
(2) receive messages from other principals,
(3) perform computations, in accordance with their policies, on the basis of (1), (2), and their current knowledge,
(4) update their knowledge, their policies, and the information in some of their datasources,
(5) send messages to other principals

The interaction in DKAL between principals takes place only via the messages mentioned in (2) and (5). The role of the underlying system is to deliver messages, with appropriate cryptographic security to provide reliable authentication of messages, which allows recipients to confirm the identity of the sender and the integrity of the message. The system may also add time stamps to messages (possibly in addition to time stamps provided by senders or recipients). Apart from this communication mechanism, all the work in a DKAL system is done locally by the principals. The possibility of environmental intervention in a principal's work will be addressed later in this report.

1.1. **Datasources.**

Datasources are similar to databases though they may be more liberal as far as the structure of the data is concerned. A datasource may be relational or XML-based or it may be a text document or any other place to store and access data. What is required is only that each datasource can be accessed via a well-defined interface that determines how queries are to be posed and how updates are to be entered.

Among the datasources, there is one that we call the *basic* datasource. Its role is to make arithmetical facts available to all principals. All principals have read-only access to it; it is never modified. (In future versions of DKAL, the role of the basic datasource may be expanded beyond arithmetical information, to include other facts that don't change and that should be available to all principals.)

We assume that, except for the basic datasource, every datasource is associated with (or "owned by") a unique principal. A principal can directly read from and write to its own datasources, but access to another principal's datasources requires communication with that other principal. In particular, if several principals wish to write to the same datasource, any write-conflicts that arise would be handled by the policy of that datasource's owner. That policy could also limit other principals' read access to the datasource, so that some principals might be able to use only a part of a datasource while other parts are accessible to more privileged principals.

Remark 1. The current implementation of DKAL does not enforce the requirement that each non-basic datasource must have a single owner. It allows shared datasources and does not yet address the issue of write conflicts. For the time being, write conflicts are resolved by the server storing the datasource.

At an abstract level, we can view all of a principal's datasources together as a (many-sorted) structure in the sense of mathematical logic ([14]) or (equivalently) a state in the sense of abstract state machines (ASMs, [9]). As in earlier versions of DKAL, we refer to this structure as the principal's *substrate*. It consists of sets of entities of certain specified types, together with relations and functions defined on these entities. Relational databases fit this description by definition, but other kinds of datasources admit easy, natural representations in this form as well.

1.2. **Communication.** Principals communicate by sending each other items of information, called *infons*, sometimes with additional data attached. Infons often look like formulas in first-order logic, but their intended meaning is quite different. It may not make sense to ask whether an infon is true or false. The meaningful question is whether it is *known* to a principal.

Example 1. We shall consider the following infon, transmitted from Bob to Alice.

(1) *bob* `said` "The Godfather" is a good movie

Alice may have the following rule:

```
with M : String, P : Principal
  upon
    P said M is a good movie
  do
    learn P said M is a good movie
```

The incoming infon will therefore match the rule condition and it will be added to Alice's knowledge base.

A principal might know some infons from the beginning of its existence; we sometimes say that the principal is *born* with this knowledge.

Example 2. Alice may trust Bob on his knowledge about good movies. If that is the case, her knowledge base should include the following infon.

(2) $(\forall M : \texttt{String})$

$$\big((\textit{bob}\ \texttt{said}\ M \text{ is a good movie}) \ \rightarrow\ M \text{ is a good movie}\big)$$

Alternatively, Alice's trust in Bob's movies taste can be encoded as a rule in her policy as follows.

```
with M : String
  if
    bob said M is a good movie
  do
    learn M is a good movie
```

Knowledge acquired at birth is particularly common when a principal is created by another principal, as an auxiliary agent. Although the current version of DKAL does not provide a mechanism for principals to create other principals and endow them with some initial knowledge and policies, such a mechanism is expected to be added in a future version of DKAL. This initial knowledge and policy may be quite small, possibly just telling the newly created principal to go to certain sources for additional knowledge and policy (and to trust those sources for this material).

Apart from this initial knowledge, infons become known to principals in accordance with their policies, typically as a result of communications or of deduction from previously known infons. But a policy rule may be triggered also by a state change (not only by an incoming message), and the execution of such a rule may also result in acquiring new knowledge. (Some principals, for example governments, can change

their knowledge or policies without any basis in pre-existing policy or communications. Such changes are outside the scope of DKAL. They have to be treated like interactions with an external environment — something happened but for no visible reason.)

Example 3. Using what she learned from communication (1) and her trust information (2), Alice can derive the following infon.

(3) "The Godfather" is a good movie

The possibility of learning infons via communication does not mean that a principal who receives an infon from another will thereby necessarily come to know that infon. Rather, a principal's policy must tell it what to do with incoming infons; some could be simply accepted (and become part of the recipient's knowledge), others could be rejected (having no effect on the recipient), and yet others could trigger any of a great variety of actions, including coming to know a different infon, updating the substrate, and sending messages. The precise description of how to react to various sorts of incoming infons is a large part of a principal's policy, and providing the means for such descriptions is a large part of the job of DKAL.

1.3. **Evidence.** A principal's policy for dealing with incoming infons may also take into account other parts of the message. In particular, infons may be accompanied by *evidence*, which would ordinarily lead to acceptance of the infon. (See below for more about "ordinarily".)

DKAL envisions two sorts of evidence. The first is a cryptographic signature of (a hash of) the infon and perhaps accompanying data. More precisely, the signature of a principal p constitutes evidence for infons of the form "p said α" or, more generally, of the form "$\beta \to (p \text{ said } \alpha)$". We refer to such infons as *speech* infons (of p). Ordinarily, a principal who receives such a signed speech infon would learn that infon.

Example 4. Bob's signature of (a hash) of infon (1) constitutes valid evidence for it.

On the other hand Bob's signature is not valid evidence for infon (3), since it is not a speech infon of his.

The second sort of evidence for an infon α would be a formal, logical deduction of α from signed speech infons and infons $\texttt{asInfon}(q)$ whose query refers to the basic datasource. A principal who receives such a deduction of α from hypotheses known (or provided) to it would ordinarily learn α.

In which logical system should these formal deductions be carried out? DKAL itself is neutral on this issue; it can work with any logical system that the principals agree on. See Subsection 1.6 for more on the issue of agreement among principals, and see Subsection 2.2 for more about logical systems and deductions.

The current implementation of DKAL uses a particular logic, primal infon logic, chosen for practical reasons: It is adequate for current applications, and it admits in practice efficient decisions about deducibility and constructions of proofs. (The worst-case complexity, though, is exponential time.) The implementation is, however, modular, so that another logical system could easily be used once one has programs for finding deductions of proposed infons from proposed hypotheses (or determining that they have no deduction).

Remark 2. Current DKAL allows but does not require messages to include evidence. Evidential DKAL, a version of DKAL described in [5], requires evidence for all messages. Future versions of DKAL may work with other conventions regarding evidence. One possibility is that certain datasources (in addition to the basic one(s)) may be designated as trustworthy, so that `asInfon` infons referring to these datasources would be allowed as premises in deductions.

1.4. **Ordinary Behavior.** In the discussion of evidence, we indicated how a principal would ordinarily react when it receives messages with appropriate evidence. There is, however, no requirement that principals actually react in the ordinary way. A principal's policy could call for rather different behavior. What, then, is the purpose of singling out one behavior as "ordinary"?

In applications, it often happens that the behavior of principals is subject to some laws or conventions, and one is interested in understanding what can happen when these laws and conventions are obeyed — in particular showing that then the whole system obeys some desirable properties, such as preservation of privacy, giving (only) appropriate principals access to resources, etc. Furthermore, if these desirable properties fail, one wants to be able to assign the blame to the particular principals that violated the laws or conventions. For this purpose, one would prove theorems, about a system of principals and their policies, saying that as long as the policies produce ordinary, law-abiding behavior, the desired properties will hold.

Thus, the word "ordinary" will be used as a shorthand to indicate the hypotheses about correct individual behavior that are needed to support conclusions about desirable properties of the entire system.

1.5. **Rounds.** The actions of any principal occur in discrete steps, which we call the *rounds* of the principal's activity. During any round, the principal has access to its substrate and its incoming messages as they were at the start of the round. In particular, messages that arrive later during a round are taken into account only in the next round. The principal performs computations during the round and decides on actions of the following sorts:

- The content of the principal's substrate may be updated.
- The principal may learn (i.e., add to its knowledge base) additional infons.
- The principal may delete infons from its knowledge base.
- The principal may send messages to other principals.
- The principal may modify its policy.

All the actions decided upon during a round are executed only at the end of the round.

More precisely, the actions are executed in parallel at the end of the round, provided they are consistent with each other. In the current implementation of DKAL, inconsitent updates of the infostrate cause the principal to simply halt, while inconsitent updates of the substrate result in an unspecified one of those updates being executed. Future implementations will incorporate more sophisticated error-handling.

Not only are the principal's own actions carried out only at the end of a round, the same also goes for interventions of an external environment. The environment can modify a principal's datasources, knowledge, and policy, but only between rounds. We have hereby adopted the position that emergency situations, which may modify things during a round, are outside the scope of DKAL (at least for the time being).

1.6. **Communities.** A DKAL *community* is a collection of principals, interacting with each other in a DKAL system, usually in connection with some common purpose.

In the traditional example of a pharmaceutical trial, all the principals involved in the trial (including the organizer, the sites, the physicians, etc.) would constitute a community in this sense.

A single principal could be in several DKAL communities, provided its activities in different communities are independent of each other. For example, a physician could participate in trials of several drugs.

Each DKAL community has certain conventions related to the operation of DKAL. In particular, it has

- a specific cryptographic system, to be used for signing and authenticating messages between its principals,
- a specific logic to be used in the deductions that principals send each other as evidence for infons,
- more generally, conventions for what constitutes ordinary behavior, and
- a specific basic datasource. (Future versions of DKAL may allow for several basic datasources in a single community.)

We assume that these community conventions are fixed once and for all; they do not change in the course of the system's operation.

2. Infon Logic

2.1. **Infons.** As indicated earlier, infons are items of information that can be known by principals and can be sent from one principal to another. The syntax of infons resembles, in certain respects, a fragment of first-order logic, but it also contains some constructions specific to DKAL. We begin our description with the more familiar part, the part that resembles multi-sorted first-order logic.

Infon logic is a typed system, but different DKAL principals may have different types as well as different function and relation symbols in their infon vocabularies. Communication works only when the symbols in a message are common to the infon vocabularies of the producer[1] and the recipient of the message, but a principal may also have, for its own use, symbols in its infon vocabulary that are not used by other principals.

All principals should have, in their infon vocabularies, the type "principal", interpreted (not surprisingly) as referring to the principals of the DKAL community. Being (or using) computers, they would ordinarily also have other standard data types, like "boolean", "integer", and "string". The current version of DKAL supports the .NET basic types, in addition to DKAL-specific types like "principal".

In addition to types, a principal's infon vocabulary contains function and relation symbols, just as in first-order logic. These relation symbols are often referred to as infon relation symbols, to emphasize that they are used to form infons, not statements. Each function and relation

[1]The producer of a message might not be the sender. For example, a principal p might simply forward a signed message that it received from another principal q. In this situation, the sender p of the forwarded message might not "understand" the message at all, though one hopes that the original author and signer of the message does understand it. Here "understand" simply means having the relevant symbols in its vocabulary.

symbol has an arity, specifying not only the number of arguments that it takes but also the types of these arguments; in addition, for function symbols, it specifies the type of the value of the function.

Terms are built as usual, from function symbols (including constant symbols, which we regard as 0-ary function symbols) and variables, respecting the types. Thus, if f is a function symbol with argument types $\tau_1, \ldots, \tau_n$ and value type τ, and if $t_1, \ldots, t_n$ are terms of types $\tau_1, \ldots, \tau_n$, respectively, then $f(t_1, \ldots, t_n)$ is a term of type τ. In this connection, the types of variables will always be given beforehand by declarations (described below).

Infon relation symbols are used to produce atomic infons, just as relation symbols are used to produce atomic formulas in first-order logic. That is, if R is a relation symbol with argument types $\tau_1, \ldots, \tau_n$, and if $t_1, \ldots, t_n$ are terms of types $\tau_1, \ldots, \tau_n$, respectively, then $R(t_1, \ldots, t_n)$ is an atomic infon.

Example 5. Infon (3) is an atomic infon constructed from the unary infon relation *is a good movie* and the constant "The Godfather".

DKAL has an additional form of atomic infons, namely `asInfon`(q), where q is a substrate query. Recall that a principal's substrate consists of datasources, each of which has a specified interface that includes asking yes-no questions. In the `asInfon` construction, q should be a query of this sort, to a specified datasource. In more detail, the syntactic form of q is $\{|a|b|\}$ where a is a string naming one of the principal's datasources and b is the actual query, whose syntax depends on the interface of the datasource named by a. The intended meaning of `asInfon`$(\{|a|b|\})$, for a ground query b, is that the principal sends the query b to the datasource a and, if the reply is affirmative, learns the infon `asInfon`$(\{|a|b|\})$. Thus, the `asInfon` constructor is a way to move information from a principal's substrate to its knowledge. It is the first of the two DKAL-specific infon constructors that have no analog in first-order logic. Usually, an `asInfon` infon would be used locally by a principal. If it is communicated to another principal, and it does not refer to a basic datasource, then it should at least be meaningful to the principal that ultimately uses it. This means that it is possible for a principal to send `asInfon`$(\{|a|b|\})$ even when it does not own a datasource named a. In particular, such an infon could occur as the antecedent of an implication. The intent would usually be that the recipient of the message has access to a datasource a and can ask it the query b. More complicated scenarios are also possible; for example, the intent might be that the recipient of `asInfon`$(\{|a|b|\})$ is expected to forward the message to yet another principal. In general, all that

is required is that, when (or if) it becomes necessary for a principal to actually evaluate `asInfon`($\{|a|b|\}$), that principal has a datasource a so that the evaluation can be carried out.

Remark 3. This requirement for an `asInfon` infon in a message is more liberal than the requirement for a premise in a deduction sent as evidence for an infon. As indicated earlier, if such a premise has the form `asInfon`(q), then the query q must refer to the basic datasource.

Relational databases, XML databases, and many other datasources admit queries containing variables and return a list of all assignments, of values to variables, that satisfy the query. A principal issuing such a query q would then learn `asInfon`(q') for all the ground instances q' of q given by the assignments returned by the query. In particular, if the query returned no instances, the principal would learn no new infons. For many sorts of datasources, the principal could, in such a case, continue by asking a negative form of the query and thus learn an infon containing the information that no assignments matched the query q.

In principle, the syntax of infons could include all the connectives and quantifiers of first-order logic. The present version of DKAL uses only conjunction ($\wedge$), weak forms of disjunction ($\vee$) and implication ($\rightarrow$), true and false ($\top$ and $\bot$, regarded as a 0-ary connectives), and universal quantification ($\forall$); furthermore, $\forall$ is not allowed to appear in the scope of any of the other constructors. Thus, infons currently have the form

$$(\forall x_1 : \tau_1) \ldots (\forall x_n : \tau_n)\, \alpha(x_1, \ldots, x_n, y_1, \ldots, y_k),$$

where α is built from atomic infons by means of $\wedge$, $\vee$, $\rightarrow$, $\top$, $\bot$, and the `said` constructor described below. Here the τ_i are types; they serve to declare the types of the quantified variables x_i. The variables y_j in α are free in the displayed infon. Infons with free variables occur only in policy rules, and their types are declared at the beginning of the policy rule in which they occur (see Subsection 3.1 below). Infons in messages or in a principal's knowledge base never have free variables. Even if, for brevity, we sometimes write such infons with apparently free variables, they are implicitly understood to be governed by universal quantifiers over the whole infon.

The one remaining DKAL-specific constructor is `said`, used in the context p `said` α, where p is a term of type principal and α is a quantifier-free infon. Infons of this form, as well as the slightly more

general form $\beta \to p$ `said` α, are called speeches of p and can be justified by the cryptographic signature of (the principal denoted by the term) p.

Example 6. Infon (1) is a speech infon of Bob, constructed using the `said` constructor on the atomic infon (3).

2.2. **Logic.** It is argued in [1] that the appropriate use of the propositional connectives and quantifiers in infon logic should be exactly intuitionistic first-order logic. This logic is, unfortunately, computationally quite complex [15].[2] Even its propositional part is polynomial-space complete, and addition of the quantifiers makes it, like classical first-order logic, undecidable.

It is therefore reasonable to seek weaker logics, rich enough for applications, but less computationally demanding. One such logic, the one currently implemented in DKAL, is primal infon logic with variables, which we describe in detail below. Its propositional fragment is decidable in linear time [12, 7]. The full logic is exponential-time in the worst case, but it seems to work quite efficiently in practice.

It might be useful, in some situations, to consider logics intermediate between primal and intuitionistic logic. For example, primal logic augmented with the rule of transitivity of implication was studied in [6]. Going further in this direction, one might allow all of intuitionistic propositional logic but restrict quantification to $\forall$ outside the scope of connectives (as in primal logic). Another possible extension would be adding variables of type infon (which can be regarded as producing a sort of second-order logic).

2.3. **Primal Infon Logic With Variables.** We turn now to a detailed description of primal infon logic. The syntax has variables, constants (but no function symbols other than constants), infon relation symbols, `said`, $\top$, $\wedge$, $\vee$, $\to$, and $\forall$. There are also punctuation symbols, namely parentheses and the colon (used for type declarations).

Terms are simply constants or variables.

Atomic infons can be built in the usual way from relation symbols and terms, but there can be other kinds of atomic infons as well. Current DKAL has the `asInfon` infons described above. The presence or absence of such additional atomic infons will make no difference to the primal infon logic, because it is not concerned with the details of atomic infons but only with the way compound infons are built using connectives, quantifiers, and `said`.

[2]By the complexity of a logic, we mean the complexity of its derivability problem: Given a set $\mathcal{H}$ of hypotheses and a proposed conclusion C, is C derivable from $\mathcal{H}$?

Compound infons are produced from atomic infons and $\top$ by applying $\wedge$, $\vee$, $\rightarrow$, and p `said` to *quantifier-free* infons and by applying $(\forall x : \tau)$ to arbitrary infons; here p is a term of type principal, and τ is a type.

To formulate the axioms and rules of inference, we adopt two notational conventions. First, we use `pref` (short for "quotation prefix") to abbreviate any expression of the form

$$p_1 \texttt{ said } p_2 \texttt{ said } \ldots p_n \texttt{ said },$$

where the p_i are terms of type principal (and where n may be zero, so that the prefix may be empty). Thus, if α is an infon without quantifiers, so is `pref` α, obtained from α by an n-fold nested application of the `said` construction. In a rule of inference where we write `pref` several times, it is understood that all these occurrences stand for the same quotation prefix.

Our second notational convention is that we omit quantifiers in presenting the axioms and rules. Thus, the infons exhibited in the following rules may contain free variables, which are to be understood as universally quantified at the beginning (not in the scope of any connective or `said`). The order of the universal quantifiers is irrelevant and we ignore it; if it were taken into account, then our deductive system should include rules to permute the quantifiers.

Notice that, according to this convention, our deductive system deals exclusively with *closed* infons, i.e., infons without free variables. Whatever might look like a free variable is actually universally quantified. We remark, in this connection, that the infons in a principal's knowledge base and the infons sent by a principal in accordance with its policy will always be closed infons. Infons with free variables play only an auxiliary role in DKAL deductions, as sub-expressions of the closed infons that are really used.

With these conventions, the deductive system is as follows.

- Axioms (or introduction rule for $\top$, with no premises)

$$\texttt{pref}\,\top$$

- Introduction rule for conjunction

$$\frac{\texttt{pref } \alpha \qquad \texttt{pref } \beta}{\texttt{pref } (\alpha \wedge \beta)}$$

- Elimination rules for conjunction

$$\frac{\texttt{pref } (\alpha \wedge \beta)}{\texttt{pref } \alpha} \qquad\qquad \frac{\texttt{pref } (\alpha \wedge \beta)}{\texttt{pref } \beta}$$

- Introduction rules for disjunction

$$\frac{\texttt{pref}\ \alpha}{\texttt{pref}\ (\alpha \vee \beta)} \qquad \frac{\texttt{pref}\ \beta}{\texttt{pref}\ (\alpha \vee \beta)}$$

- Introduction rule for implication

$$\frac{\texttt{pref}\ \beta}{\texttt{pref}\ (\alpha \to \beta)}$$

- Elimination rule for implication

$$\frac{\texttt{pref}\ \alpha \qquad \texttt{pref}\ (\alpha \to \beta)}{\texttt{pref}\ \beta}$$

- Quantifier rule

$$\frac{\texttt{pref}\ \alpha(x)}{\texttt{pref}[t/x]\ \alpha(t)},$$

where t is a term of the same type as the variable x, and where $\texttt{pref}[t/x]$ means the result of replacing x by t in the quotation prefix pref.[3]

To clarify the nature of this deductive system, we compare it with the standard natural-deduction rules for $\top$, $\wedge$, $\vee$, $\to$, and $\forall$, and we comment on the differences.

The first difference is that all the propositional rules of primal logic operate within the scopes of the quotation prefixes and the unwritten quantifiers. The presence of quotation prefixes incorporates into the deductive system some conventions about the intended meaning of "said". For example, a principal who said α and also said β is understood to have (implicitly) also said $\alpha \wedge \beta$.

Remark 4. The careful reader may be concerned that, in view of our convention that each of the premises and conclusions in our rules is to be understood as having, at the beginning, universal quantifiers for all its free variables, it is possible that, in a particular application of a rule, the quantifiers in the various premises and the conclusion govern different variables. This is because different variables may be free in the premises and in the conclusion. Consider, for example, a conjunction elimination rule

$$\frac{P(x) \wedge Q(y)}{P(x)}.$$

Written out in full, with the implicit quantifiers made explicit, this is

$$\frac{\forall x \forall y\, (P(x) \wedge Q(y))}{\forall x\, P(x)},$$

[3]If the type of x is not "principal" then x cannot occur in a quotation prefix, so $\texttt{pref}[t/x]$ is simply pref.

where y is quantified in the premise but not in the conclusion.

If desired, we could avoid such differences by adopting the convention that, in an instance of a deduction rule, all the premises and conclusion should be governed by universal quantifiers of all variables that occur free anywhere in the instance. In the example above, we would then have $\forall x \forall y\, P(x)$ in the conclusion. We would then have to add to the system rules for inserting and removing dummy quantifiers, i.e., quantifiers whose variables don't occur free in their scopes, like the $\forall y$ in $\forall x \forall y\, P(x)$. In this example, we would need to infer the original conclusion, $\forall x\, P(x)$ from the new conclusion $\forall x \forall y\, P(x)$. (We assume all types are nonempty, so that removing a dummy quantifier is semantically sound.)

The second, perhaps more surprising, difference from intuitionistic natural deduction is that the implication introduction rule is very weak. Intuitionistic logic would allow one to infer $\alpha \to \beta$ if one had deduced β from α, but primal logic allows that inference only if one has proved β outright. (In particular, the deductive system would be sound for an interpretation in which $\alpha \to \beta$ means simply β.) The inference from β to $\alpha \to \beta$ is not as useless as it might appear at first sight. A principal that knows β might nevertheless want to tell some other principal only the implication $\alpha \to \beta$. An advantage of primal logic is that the weakened rule permits far more efficient algorithms for the decision problem, yet has proved adequate for actual applications.[4]

Remark 5. Datalog can be regarded as a rather restricted fragment of primal infon logic. Specifically, if we remove from primal infon logic the quotations, disjunction (and its introduction rules), conjunction elimination, and implication introduction rules (retaining only the conjunction introduction, implication elimination, and quantifier rules), the logical system that results is essentially Datalog. More precisely, since Datalog does not use explicit quantifiers, our rules should, for comparison with Datalog, be read literally as written, not with the quantifiers that primal infon logic tacitly reads. In particular, our quantifier rule then amounts to substitution for free variables, which is part of the inference mechanism of Datalog.

[4]Note that our change in the implication introduction rule removes a place where the intuitionistic rules for our connectives would allow discharging (also called canceling) hypotheses.

A third difference is that we have only introduction rules for disjunction, with no elimination rule.[5] As with the weakened rule for implication introduction, the justification for this omission is computational efficiency combined with adequacy for applications.

A fourth difference from intuitionistic natural deduction concerns the quantifier rule(s). We have nothing like the intuitionistic $\forall$-introduction rule,

$$\frac{\alpha(x)}{\forall x\, \alpha(x)},$$

which allows free variables to be universally quantified (provided they are not also free in any undischarged hypotheses). Such a rule would be superfluous in our system because all the variables that seem to be free are, according to our convention, really already universally quantified.

Our quantifier-elimination rule looks, at first glance, like the $\forall$-elimination rule of intuitionistic logic, but there are differences. Because of our convention of implicit quantification, this rule not only eliminates a quantifier (the implicit quantifier of x in the premise) but might also introduce other quantifiers (implicit quantifiers of variables occurring in t in the conclusion). Also, we do not impose the usual requirement that no variable in t be "accidentally" quantified by quantifiers within α. We don't need such a requirement because the only quantifiers that could "catch" variables in t would be our implicit universal quantifiers at the beginning of α, and those quantifiers are ones we (implicitly) want to have governing those variables anyway.

We use in primal infon logic the notion of deduction familiar from many other logical systems. Thus, a *deduction* of a closed infon α from a set Γ of closed infons is a finite sequence of closed infons, ending with α, such that each infon in the sequence is either a member of Γ or an axiom or a consequence of earlier infons in the sequence via one of the rules of inference. For use in messages, we specify the syntactic form of a deduction as the concatenation of the infons in the sequence, separated by commas, with pointers indicating, for each infon that results from application of a rule of inference, the premise(s) of that rule.

3. Policies

A principal's policy is a set of policy rules, telling what to do in any round, on the basis of the messages received and the principal's

[5]Again, we omit a rule that would discharge hypotheses. Because of this omission and the weakening of the implication introduction rule, primal infon logic with variables is a Hilbertian system in the sense of [4].

knowledge, as of the beginning of the round. The possibilities for what to do include modifying the principal's knowledge base by adding or removing infons, modifying the datasources in the principal's substrate, modifying the principal's policy, and sending messages.

A policy rule thus contains two major parts: *Conditions* tell under what circumstances the rule is to be executed. *Actions* tell what is to be done. In addition to these two major parts, a rule also has, at the beginning, *declarations* giving the types of any variables free in the rule. (Variables bound later within the rule can only occur inside infons, are universally quantified in those infons, and have their types declared at the point where they are quantified.)

Most of this section is devoted to explaining the various sorts of conditions and actions that are available in DKAL (or are expected to become available), but first we take care of the overall structure of rules and declarations.

3.1. **Rules and Declarations.** A declaration of a variable x has the form $x : \tau$, where τ is a type. A rule begins, in general, with a *declaration block*, whose syntax is the keyword "`with`" followed by a finite sequence of declarations of distinct variables, i.e.,

$$\texttt{with}\ x_1 : \tau_1,\ x_2 : \tau_2,\ \ldots\ , x_k : \tau_k.$$

The sequence may have length zero, in which case the word `with` may be omitted. The order of the declarations within the block is irrelevant. The variables declared in a rule's declaration block will be called the *rule-wide* variables of the rule. They can occur anywhere in the rule, in contrast to other variables, which can occur only within the infons where they are quantified.

A rule consists in general of the following items, in order:

- a declaration block,
- some conditions,
- the keyword "`do`',"
- some actions.

Rule-wide variables can occur in the conditions and the actions. The conditions will determine which, if any, assignments of values to these variables verify the conditions. The actions will then be executed for just those assignments.

Conditions and actions can also contain the keyword `me`, which functions as a constant of type principal; when it occurs in the policy of a principal q, it is interpreted as denoting q.

3.2. **Conditions.** DKAL has two sorts of conditions or guards, used to indicate when a rule should be executed. One sort, using the keyword "`if`", checks whether any infon of a specified form is deducible from the principal's knowledge base. The other, using the keyword "`upon`", checks whether there is a new incoming message of a specified form. We shall now describe these conditions in detail.

The syntactic form of an `if`-condition is simply "`if` α", where α is an infon. Any variable that occurs free in α must be among the rule-wide variables of the rule. When the rule is to be applied by a principal p, these variables must be instantiated by constants in a way that makes the infon deducible from the knowledge base of p.

More formally, in the context of primal logic, we define an *instantiation* for a rule to be a function σ from its rule-wide variables to constants, respecting types (i.e., the type of the constant $\sigma(x)$ must be the same as the type declared for the variable x in the rule's declaration block).[6] For such a σ and an infon α, the result of replacing all free occurrences in α of rule-wide variables x by the corresponding constants $\sigma(x)$ will be denoted by $\sigma(\alpha)$ and called an *instance* of α. An instantiation σ is said to *verify* an infon α for a particular principal in a particular round if the instance $\sigma(\alpha)$ is deducible from the knowledge base that this principal had at the start of this round. The only instantiations that will be relevant to the execution of a rule (by a principal in a round) are those that simultaneously verify all the `if`-conditions of that rule and also the `upon`-condition, if any, which we treat next.

The simplest form of an `upon` condition is "`upon` α", where α is an infon all of whose free variables are rule-wide. It is verified by an instantiation σ for a principal in a round if and only if the infon $\sigma(\alpha)$ is among the principal's newly received messages at the beginning of that round. Here "newly received" means that the message was received by the start of the current round but not by the start of the previous round (if any).

A second sort of `upon` condition has the form "`upon` α `from` p", where α is as above and p is a term of type principal. It is verified by an instantiation σ if and only if $\sigma(\alpha)$ is a newly arrived message sent by the principal $\sigma(p)$. (Note that p could be either a constant or a rule-wide variable of type principal; in the latter case, it is instantiated by σ along with the free variables in α.)

[6] In the context of logics admitting function symbols other than constants, not only constants but also other ground terms should be allowed as values of instantiations.

Either of the preceding sorts of `upon` rules can be modified by the keyword "`justified`", so the rules can have the form

`upon justified` α or `upon justified` α `from` p.

In this case, verification by an assignment is defined as above but with the additional requirement that the newly arrived $\sigma(\alpha)$ (sent by $\sigma(p)$ in the case of `from`) must be accompanied by evidence as described in Subsection 1.3 above.

Future versions of DKAL may replace the simple "`justified`" by more specific formulations restricting the sort of evidence that is acceptable. They may also provide more relaxed notions of justification, involving reasons for belief rather than rigorous deductions.

A DKAL rule can have any finite number of `if` conditions but at most one `upon` condition. We say that an instantiation *fires* the rule if it verifies all its conditions.

When a principal evaluates the conditions in a rule, it processes them in the order in which they are written. Instantiations that result from the earlier conditions are applied to the later ones before the later ones are evaluated. In some situations, the ordering of the conditions can affect the ability of the datasources to handle `if` conditions; the reason is that a datasource might be able to handle an instantiated (or partly instantiated) form of a query that it cannot handle without the instantiation.

Example 7. Consider the following rule in Alice's policy:

```
with M : String, R : Double
  if
    asInfon {| basic | R > 4.75 |}
  upon
    bob said M is rated R stars
  do
    say to erin :
      M is a great movie
```

When the first condition is evaluated, there is no concrete value assigned to R yet. The basic substrate cannot evaluate such a query; doing so would return an infinite set of answers, one for each number greater than 4.75. Consequently, the evaluation fails.

If the two conditions were in inverse order the evaluation would be successful, as R carries a concrete value by the time the basic substrate is queried.

3.3. **Actions.** We turn next to the description of the actions that a principal can execute and the syntactic description of those actions in its policy. Syntactically, the actions will be within rules, so there will always be a well-defined list (possibly empty) of rule-wide variables; these are the only variables that can occur free in the actions. Furthermore, there will always be a well-defined set of relevant assignments of values to these variables, namely the assignments that verify the rule's conditions. The actions are to be executed, in parallel, for all these assignments.

Notice that, once an assignment is applied to an action, there will no longer be any free variables. (There may be quantified variables, in the infons that are part of the action.) Accordingly, when we describe how to execute actions, we shall assume that they contain no free variables.

There are four general categories of actions: those that update a principal's substrate, those that update its knowledge base, those that update its policy, and those that send messages. In the rest of this subsection, we describe them in order. We do not describe actions that would create new principals, because these are not yet implemented.

Actions by which a principal updates its substrate have the form "`apply` u", where u, a *substrate update*, has the form $\{|a|b|\}$ similar to substrate queries; a names the datasource that is to be updated, and b is the actual update instruction, in the syntax appropriate for that datasource.

Actions by which a principal updates its knowledge base have the forms "`learn` α" and "`forget` α". The former adds the closed infon α to the principal's knowledge base if it isn't already there; the latter removes it if it is already there. (Recall that, in the rule itself, α could contain free variables, but, when the action is executed, these variables have been instantiated by constants according to some assignment verifying the conditions of the rule. So what is learned or forgotten is always a closed infon. It may, of course, contain quantified variables, just not free ones.) It is entirely possible that the "forgotten" infon α may be derivable from other infons that remain in the principal's knowledge; see Subsection 3.4 below for more details about this point.

Actions by which a principal updates its policy have the forms "`install` R" and "`uninstall` R", where R is a rule. The former adds R to the principal's policy if it is not already there, and the latter removes it if it is already there.

Finally, there are three forms of actions that send messages. The simplest is "`send to` $p : \alpha$" where p is a principal term and α is an infon. Because of instantiation, when the action is executed, p is a constant denoting a principal, and the message is sent to this principal.

The message itself is a closed infon, obtained as the corresponding instantiation of α.

Notice that, if the infon α is a speech infon of the sender q, i.e., if it is of the form q `said` γ or $\beta \to q$ `said` γ, then the cryptographic signature automatically provided by DKAL constitutes evidence for α, so that the recipient p would ordinarily learn α. If, however, α is not of this form, then the sender q can provide evidence for α in the form of a deduction of α from (1) speech infons known or provided to the recipient p and (2) `asInfon` infons verifiable from the basic data source. The syntax for this action is "`send to` $p : \alpha$ `and derivation` d", where d is the desired derivation of α.

Finally, DKAL has the action "`say with justification to` $p : \alpha$", which, when executed by a principal q, is equivalent to "`send to` p : q `said` α". That is, it converts the given infon α to a speech infon of the sender by prepending "q `said`" and sends the result. Being a speech infon of the sender, this message is automatically justified. DKAL allows a simple "`say`" as an abbreviation for "`say with justification`".

Example 8. Alice may wish to tell her friends about any good movies that she discovered. The rule in her policy might look like this:

```
with M : String, P : Principal
  if
    M is a good movie
  if
    P is a friend of mine
  do
    say to P : M is a good movie
```

Chuck, a friend of Alice, may keep a list of movies he wishes to see in a datasource. Whenever Alice recommends a good movie, he updates his list.

```
with M : String
  upon
    alice said M is a good movie
  if
    M unwatched
  if
    asInfon {| movie-wishlist | M not listed |}
  do
    apply {| movie-wishlist | add M |}
```

3.4. **Implicit and Explicit Knowledge.** The infons in a principal's knowledge base are *explicitly* known to that principal. In addition, the principal *implicitly* knows the infons that can be deduced from its knowledge base. As indicated above, an action of the form "`forget` α" removes α from the principal's explicit knowledge, but it may leave α in the implicit knowledge, so that a condition "`if` α" would be satisfied.

It can be dangerous to ignore the distinction between explicit and implicit knowledge, as the behavior of some rules can then be quite unexpected. For example, one might think that a rule of the form

$$\texttt{if } \gamma \texttt{ then learn } \gamma$$

would have no effect; it merely tells the principal to learn something that it already knows. Here is an example where such a rule has an observable effect.

Example 9. Suppose a principal, whose logic includes at least primal logic, explicitly knows α, β, and nothing else. Now suppose it executes, in successive rounds, the two rules

(1) `do forget` α
(2) `if` α `do send to` $p : \alpha$.

Nothing will be sent to p, because, having forgotten α in the first round, the principal will not find α satisfied at the second round.

Suppose, instead, that the principal executes in successive rounds three rules, of which the first is of the form "learn what you already know" and the other two are exactly as above:

(1) `if` $\beta \to \alpha$ `then do learn` $\beta \to \alpha$
(2) `do forget` α
(3) `if` α `do send to` $p : \alpha$.

Now α will be sent to p. The reason is as follows. Since the principal initially knows α and since $\beta \to \alpha$ follows by $\to$-introduction, the first rule will fire and will put $\beta \to \alpha$ into the principal's *explicit* knowledge (whereas, before, it was only implicit). The second rule makes the principal delete α from its explicit knowledge, but $\beta \to \alpha$ remains. Finally, the third rule, executed with $\beta \to \alpha$ and β in the knowledge base, finds that α is deducible by $\to$-elimination, so it fires and sends α to p.

A possible future development of DKAL might replace the current implementation of "`forget` α" with one that deletes not only α but any other explicit knowledge that was obtained on the basis of α. Thus, in the preceding example, it would delete $\beta \to \alpha$, and the two-rule and three-rule scenarios would agree.

Of course such an extension of the `forget` action would require principals to keep logs of the reasons why infons were added to their knowledge bases. Furthermore, ambiguities could arise if several reasons could have led to the same action. Here is an example, using a logic with disjunction.

Example 10. Suppose a principal has α, β, and nothing else in its knowledge base, and suppose it executes, in successive rounds, the three rules

(1) `if` $\alpha \vee \beta$ `then do learn` γ
(2) `do forget` α
(3) `if` γ `then do send to` $p : \gamma$.

When executing the first rule, this principal has two ways to deduce γ from its knowledge, namely the disjunction introduction rules using either α or β as the premise. If it chooses the deduction using α, then in the next round, when it forgets α, it will also delete γ from its knowledge base, because γ was added on the basis of a deduction using α. Therefore, when it reaches the third rule, it will find the condition γ unsatisfied, so it will send nothing. If, on the other hand, when executing the first rule, it chooses the deduction using β, then forgetting α does no damage to γ, and the third rule will result in γ being sent to p.

Acknowledgments. We thank Artem Melenyev and Nikhil Swamy for useful discussions of various aspects of DKAL. Melentyev also provided the most recent updates of the implementation of DKAL.

References

[1] Lev Beklemishev, Andreas Blass, and Yuri Gurevich, "The logic of infons," in preparation.
[2] Lev Beklemishev and Yuri Gurevich, "Propositional primal logic with disjunction," Microsoft Research Technical Report MSR-TR-2011-35 (March, 2011).
[3] Andreas Blass and Yuri Gurevich, "Abstract state machines capture parallel algorithms," *A. C. M. Trans. Computational Logic* 4 (2003) 578–651; Correction and extension, *ibid.* 9 (2008) Article 19.
[4] Andreas Blass and Yuri Gurevich, "Hilbertian deductive systems, infon logic, and Datalog," *Bull. European Assoc. Theoret. Comp. Sci.* 102 (October, 2010) 122–150; revised version Microsoft Research Technical Report MSR-TR-2011-81 (June, 2011).
[5] Andreas Blass, Yuri Gurevich, Michał Moskal, and Itay Neeman, "Evidential authorization," in *The Future of Software Engineering,* ed. S. Nanz, Springer-Verlag (2011) 77–99.
[6] Carlos Cotrini and Yuri Gurevich, "Transitive Primal Infon Logic: the Propositional Case," Microsoft Research Technical Report MSR-TR-2012-15

[7] Carlos Cotrini and Yuri Gurevich, "Basic primal infon logic," Microsoft Research Technical Report MSR-TR-2012-88
[8] DKAL web site at CodePlex, `http://dkal.codeplex.com`
[9] Yuri Gurevich, "Evolving algebra 1993: Lipari Guide," in *Specification and Validation Methods* ed. E. Börger, Oxford Univ. Press (1995) 9–36.
[10] Yuri Gurevich and Itay Neeman, "DKAL: Distributed knowledge authorization language," Microsoft Research Technical Report MSR-TR-2007-116 (August, 2007); revised version MSR-TR-2008-09 (January, 2008); extended abstract in *21st IEEE Computer Security Foundations Symposium (CSF 2008)* 149–162.
[11] Yuri Gurevich and Itay Neeman, "DKAL 2 — A simplified and improved authorization language," Microsoft Research Technical Report MSR-TR-2009-11, (February, 2009).
[12] Yuri Gurevich and Itay Neeman, "Infon logic: The propositional case," *Bull. European Assoc. Theoret. Comp. Sci.* 98 (June, 2009) 150–178; revised version *ACM Trans. Comp. Logic* 12:2 (January, 2011) article 9.
[13] Yuri Gurevich and Arnab Roy, "Operational semantics for DKAL: Application and analysis," in *TrustBus 2009, 6th International Conf. on Trust, Privacy and Security in Digital Business*, Springer-Verlag Lecture Notes in Computer Science 5695 (2009) 149–158.
[14] Joseph Shoenfield, *Mathematical Logic*, Addison-Wesley (1967).
[15] Richard Statman, "Intuitionistic propositional logic is polynomial-space complete," *Theoret. Comp. Sci.* 9 (1979) 67–72.

Mathematics Department, University of Michigan, Ann Arbor, MI 48109–1043, U.S.A.
E-mail address: `ablass@umich.edu`

Computer Science Department, FCEyN, University of Buenos Aires, Buenos Aires (C1428EGA), Argentina
E-mail address: `gdecaso@dc.uba.ar`

Microsoft Research, One Microsoft Way, Redmond, WA 98052
E-mail address: `gurevich@microsoft.com`