A Formal Study of Collaborative Access Control in Distributed Datalog

Serge Abiteboul[1], Pierre Bourhis[2], and Victor Vianu[3]

1 INRIA Saclay, Palaiseau, France; and
ENS Cachan, Cachan, France
serge.abiteboul@inria.fr
2 CNRS, France; and
Centre de Recherche en Informatique, Signal et Automatique (CRIStAL) – UMR 9189, Lille, France
pierre.bourhis@univ-lille1.fr
3 University of California San Diego, La Jolla, USA; and
INRIA Saclay, Palaiseau, France
vianu@cs.ucsd.edu

Abstract

We formalize and study a declaratively specified collaborative access control mechanism for data dissemination in a distributed environment. Data dissemination is specified using distributed datalog. Access control is also defined by datalog-style rules, at the relation level for extensional relations, and at the tuple level for intensional ones, based on the derivation of tuples. The model also includes a mechanism for "declassifying" data, that allows circumventing overly restrictive access control. We consider the complexity of determining whether a peer is allowed to access a given fact, and address the problem of achieving the goal of disseminating certain information under some access control policy. We also investigate the problem of information leakage, which occurs when a peer is able to infer facts to which the peer is not allowed access by the policy. Finally, we consider access control extended to facts equipped with provenance information, motivated by the many applications where such information is required. We provide semantics for access control with provenance, and establish the complexity of determining whether a peer may access a given fact together with its provenance. This work is motivated by the access control of the Webdamlog system, whose core features it formalizes.

1998 ACM Subject Classification H.2.0 [Database Management] General – Security, integrity and protection, H.2.1 [Database Management] Logical Design – Data models, H.2.3 [Database Management] Languages – Data description languages, data manipulation languages

Keywords and phrases Distributed datalog, access control, provenance

Digital Object Identifier 10.4230/LIPIcs.ICDT.2016.10

1 Introduction

The personal *data* and favorite *applications* of Web users are typically distributed across many heterogeneous devices and systems. In [19], a novel *collaborative access control mechanism* for a distributed setting is introduced in the context of the language Webdamlog, a datalog-style language designed for autonomous peers [3, 2]. The experimental results of [19] indicate that the proposed mechanism is practically feasible, and deserves in-depth investigation. In the present paper, we provide for the first time formal grounding for the mechanism of [19] and answer basic questions about the semantics, expressiveness, and computational cost of such a mechanism. In the formal development, we build upon *distributed datalog* [16, 20], which abstracts the core of Webdamlog, while ignoring certain features, such as updates and delegation.

In this investigation, as in Webdamlog, access control is *collaborative* in the following sense. The system provides the *means* to specify and infer access rights on disseminated information, thus *enabling* peers to collectively enforce access control. The system is agnostic as to how peers are motivated or coerced into conforming to the access control policy. This can be achieved in various ways, from economic incentives to legal means (see, e.g., [28]), possibly relying on techniques such as encryption or watermarking (see, e.g., [5]). We do not address these aspects here.

The access control of [19] that we formalize and study here works as follows. First, each peer specifies which other peers may access each of its extensional relations using *access-control-list rules.* This provides in a standard manner an initial coarse-grained (relation-at-a-time) access control, enforced locally by each peer. Next, facts can be derived among peers using *application rules.* Access control is extended to such facts based on their provenance: to see a propagated fact, a peer must have access to the extensional relations used by the various peers in producing the fact. This enables controlling access to data *disseminated* throughout the entire network, at a fine-grain (i.e., tuple) level. This capability is a main distinguishing feature of Webdamlog's access control model. The access control also includes a *hide* mechanism that allows circumventing overly restrictive access control on some disseminated facts, thus achieving a controlled form of "declassification" for selected peers.

Access control in distributed datalog raises a variety of novel semantic, expressiveness and complexity issues. How complex is it to check whether a peer has the right to access a propagated fact? What are the appropriate complexity measures in this distributed setting? Does the access control mechanism prevent leakage of unauthorized information? What does it mean to extend access control to facts equipped with their *provenance*? Is there an additional cost? These are some of the fundamental questions we study, described in more detail next.

While the experimental results of [19] suggest that the computational cost of the proposed mechanism is modest, we show formally that its complexity is reasonable. Specifically, we prove that the data complexity of determining whether a peer can access a given fact is PTIME-complete (with and without *hide*).

We next consider the problem of information leakage, which occurs when a peer is able to infer some facts to which the peer is not allowed access by the policy. We show that, while undecidable in general, information leakage can be tested for certain restricted classes of policies and is guaranteed not to occur for more restricted classes.

One of the challenges of access control is the intrinsic tension between access restrictions and desired exchange of information. We consider the issue of achieving the goal of disseminating certain information under some access control policy. The goal is specified as a distributed datalog program. We show that it is undecidable whether a goal can be achieved without declassification (i.e., without *hide*). We study the issue of finding a policy without *hide* that achieves a maximum subset of the specified goal. While any goal can be achieved by extensive use of *hide*, we show, more interestingly, how this can be done with *minimal* declassification.

In many applications, it is important for inferred facts to come with *provenance* information, i.e., with traces of their derivation. We demonstrate that adding such a requirement has surprising negative effects on the complexity. For this, we introduce an intermediate measure between data and combined complexity, called *locally-bounded* combined complexity that allows making finer distinctions than the classical measures in our context. The intuition is that the peers are seen as part of the data and not of the schema, which is more in the spirit of a Web setting. We show that the locally bounded complexity of query answering increases from PTIME-complete to PSPACE-complete when it is required that the query answer carries *provenance* information.

The organization is as follows. Section 2 recalls the distributed datalog language [3]. In Section 3, we formalize the core aspects of the access control mechanism of [19], establish the complexity of answering queries under access control. Information leakage is studied in Section 4. The issue of achieving some dissemination goal under a particular access control policy is the topic of Section 5. Access control in the presence of provenance is investigated in Section 6. Finally, we discuss related work and conclude.

2 Distributed Datalog

In this preliminary section, we formally define a variant of distributed datalog, which captures the core of Webdamlog [3].

The language. We assume infinite disjoint sets Ext of extensional relation symbols, Int of intensional relation symbols, $\mathcal{P}$ of *peers* (e.g. p, q), $\mathcal{D}_p$ of *pure data values* (e.g., a, b), and $\mathcal{V}$ of *variables* (e.g., x, y, X, Y). For relations, we use symbols such as R, S, T. The set $\mathcal{D}$ of *constants* is $\mathcal{P} \cup \mathcal{D}_p \cup Ext \cup Int$. A *schema* is a mapping σ whose domain $dom(\sigma)$ is a finite subset of $\mathcal{P}$, that associates to each p a finite set $\sigma(p)$ of relation symbols in $Int \cup Ext$, with associated arities. Let σ be a schema, $p \in dom(\sigma)$. A relation R in $\sigma(p)$ is denoted by $R@p$, and its arity by $arity(R@p)$. We denote $ext(p) = \sigma(p) \cap Ext$, $int(p) = \sigma(p) \cap Int$, $ext(\sigma) = \cup_{p \in dom(\sigma)} ext(p)$, and $int(\sigma) = \cup_{p \in dom(\sigma)} int(p)$. An *instance* I over σ is a mapping associating to each relation schema $R@p$ a finite relation over $\mathcal{D}$ of the same arity. For a tuple $\bar{a}$ in $I(R@p)$, the expression $R@p(\bar{a})$ is called a (p-)*fact* in $R@p$. An *extensional* instance is one that is empty on $int(\sigma)$. Observe that $R@p$ and $R@q$, for distinct p, q, are distinct relations with no a priori semantic connection, and possibly different arities. Note also that an expression $R@p(a_1, ..., a_k)$ for $R, p, a_1, ..., a_k$ in $\mathcal{D}$ is a *fact for* a schema σ if: p is a peer in $dom(\sigma)$, R is a relation schema in $\sigma(p)$, and $arity(R@p) = k$. Note that relations may contain pure data values, peers, as well as relation symbols. Finally, (U)CQ denotes (unions) of conjunctive queries (see [6]).

▶ **Definition 1** (distributed datalog). A *d-datalog* program P over schema σ is a finite set of rules of the form

- $Z_0@z(\bar{x}_0) \text{ :- } R_1@p(\bar{x}_1), \cdots R_k@p(\bar{x}_k)$ where
- $p \in dom(\sigma)$, $k \geq 0$, and for every $i \geq 1$, R_i is in $\sigma(p)$ and $\bar{x}_i$ is a vector of variables and constants in $\mathcal{D}$ of the proper arity;
- $z \in dom(\sigma) \cup \mathcal{V}$, $Z_0 \in Int \cup \mathcal{V}$; and
- each variable occurring in the head appears in $\bar{x}_i$ for some $i \geq 1$.

Note that the relation or peer names in the head may be variables. Note also that all the relations in the body of a rule come from the same peer. Although we define a global d-datalog program, one should think of each peer p as having its separate program consisting of all the rules whose bodies use relations at p.

▶ **Example 2.** Consider the rules:

- $Album@Alice(x) \text{ :- } Album@Bob(x)$
- $Album@z(x) \text{ :- } Album@Bob(x), Friend@Bob(z)$
- $Z@z(x) \text{ :- } Album@Bob(x), FriendPhotos@Bob(Z, z)$

Bob uses the first rule to publish his photos in Alice's album, and the second to publish his photos in all of his friends' albums (peer variable z). In the last rule, different names can be used for the relations where the friends keep their photos (variable Z for a relation name).

A d-datalog program defines the meaning of intensional relations from given extensional relations. The semantics is in the spirit of the datalog semantics. More precisely:

▶ **Definition 3** (Semantics). Let P be a d-datalog program over some schema σ. The *immediate consequence operator* Γ_P on instances over σ is defined as follows. Let I be an instance over σ.

Consider a rule $Z_0@z(\bar{x}_0)$:– $R_1@p(\bar{x}_1), \cdots R_k@p(\bar{x}_k)$ of P. An *instantiation* of the rule in I is a mapping ν from its variables to the active domain (the set of values occurring in P, I, or $dom(\sigma)$), extended with the identity on constants, such that:

- for each $i \geq 1$, $R_i@p(\nu(\bar{x}_i)) \in I$; and
- $\nu(Z_0)@\nu(z)(\nu(\bar{x}_0))$ is a fact for schema σ.

$\Gamma_P(I)$ is obtained by adding to I all facts $\nu(Z_0)@\nu(z)(\nu(\bar{x}_0))$ where ν is an instantiation in I of some rule $Z_0@z(\bar{x}_0)$:– $R_1@p(\bar{x}_1), \cdots R_k@p(\bar{x}_k)$ of P. Note that Γ_P is monotonic. The *semantics* of P for an extensional instance I, denoted $P(I)$, is the mapping associating to each extensional instance I the *projection* on the intensional relations of P of the *least fixpoint* of Γ_P containing I.

Observe that a rule may "attempt" to derive an improper fact, for which $\nu(z)$ is not in $dom(\sigma)$, or $\nu(Z_0)$ is not a relation in $\sigma(\nu(z))$, or the arity is incorrect. In such cases, the fact is simply *not* derived.

▶ Remark. Consider a rule with variable peer or relation name. Suppose for instance that both are variables. A *head-instantiation* ν of that rule for a schema σ is a mapping over Z_0, z such that $\nu(z)$ is a peer of σ, $\nu(Z_0)$ an intensional relation of $\sigma(\nu(z))$, and $arity(\nu(Z_0))$ $= |\bar{x}_0|$. One can define similarly the notion of head-instantiation for a rule with only a variable peer or only a variable relation name. It is easy to see that the program obtained by replacing each rule by all its head-instantiations has the same semantics as the original. So if the set of peers is fixed (known in advance), one can assume that, for each rule, the name of the relation and the peer in the head are constants.

3 The access control model

In this section, we formalize the core aspects of the access control mechanism of [19]. The focus here is on the READ privilege; we will ignore the GRANT privilege (allowing a peer to define permissions on another peer's relations) and the WRITE privilege (allowing a peer to push data to another peer's relations), see [19]. We also provide in this section basic expressiveness and complexity results on access control.

The extensional relations at a given peer are owned by the peer. The peer can give READ privilege on these extensional relations to other peers. This is specified at each peer p using an intensional relation $acl@p$ (for *access control list*) of arity 2. A fact $acl@p(R, q)$ states that peer q is allowed to read the extensional relation $R@p$.

In the following, we assume that for each peer p, $acl \in int(p)$ and $arity(acl@p) = 2$. For instance, a rule "$acl@p(R, z)$:– $Likes@p(z)$" can be used in a program to grant access to relation $R@p$ to all the peers z that are in relation $Likes@p$.

A d-datalog *program* P with access control (denoted d-datalog$_{ac}$) over some schema σ is a finite set of d-datalog rules $Z_0@z(\bar{x}_0)$:– $R_1@p(\bar{x}_1), \cdots R_k@p(\bar{x}_k)$, where $R_1, ..., R_k$ are not *acl* and the rules are of one of the following two kinds:

- **Application rule:** Z_0 is not *acl*; and
- **Access control rule:** The rule head is $acl@p(Z, z)$ for some terms Z, z.

Given a program P, the set of application rules forms the *application program* of P, denoted P_{app}, and the set of access control rules forms the *(access control) policy* of P, denoted P_{pol}. Facts of the form $acl@p(R, q)$ are called *access control facts*, and the others are called *application facts*. It should be noted that no such distinction is made in Webdamlog. We distinguish here between access control and application rules to be able to formally compare access control policies.

The meaning of an access control policy P_{pol} for a given extensional instance I is clear in the absence of intensional relations: use the access rules to compute at each peer the set of peers allowed to read its extensional relations. This yields relation-at-a-time, coarse-grained access control to the extensional relations. For intensional relations, we use tuple-level fine-grained access control. Intuitively, an intensional fact can be read by a peer p if it can be derived by some application of a rule from tuples that p is already allowed to access. Then, for a d-datalog$_{ac}$ program P, P_{app} and P_{pol} may interact recursively: the derivation of an intensional fact may yield some new permission for an extensional relation, which, in turn, may enable the derivation of a new intensional fact, and so on. The fine-grained access control at the tuple level is illustrated in an example.

▶ **Example 4.** Consider the program P:

$$\begin{array}{lll}
P_{pol} & acl@Bob(Album, z) & :\!-\ friends@Bob(z); \\
 & acl@Bob(Tagged, z) & :\!-\ friends@Bob(z); \\
P_{app} & Album@z(x) & :\!-\ Album@Bob(x),\ Tagged@Bob(x, z);
\end{array}$$

The access control rules allow Bob's friends access to his *Album* and *Tagged* relations. The application rule transfers to a given person the photos in which he/she is tagged. Consider a photo α with tagging *Sue*, assuming she is a friend of Bob. Then the picture α belongs (intensionally) to Sue's album. A friend of Bob who will ask to see Sue's album will see the photo α.

With standard access control, peers are only be able to control access to their local data. With the proposed mechanism, they further control the *dissemination* of their data. In other words, they can control what *other* peers should do with their data. This is achieved by propagating, together with data, permissions via application rules, based on *provenance* information about derived facts. A tuple derived by some instantiation of an application rule is accessible by a peer if that peer has access to each tuple in the body of the rule.

The semantics. To define the semantics of programs, we associate with each peer p in $dom(\sigma)$ and each relation $R@p$, $R \neq acl$, a relation $\widehat{R}@p$ of arity $arity(R) + 1$. Intuitively, $\widehat{R}@p(\bar{x}, q)$ says that peer q is allowed access to the fact $R@p(\bar{x})$. The semantics is defined using a d-datalog program. We describe next the construction of that program.

▶ **Definition 5** ($\widehat{P}$ **construction).** The semantics of a d-datalog$_{ac}$ program P over some schema σ for an extensional instance I over σ is defined using a d-datalog program $\widehat{P}$ (without access control) defined as follows. Its schema consists of: (i) the extensional and intensional relations of σ; and (ii) intensional relations $\{\widehat{R}@p \mid R@p \in \sigma(p), R \neq acl\}$.

The rules of $\widehat{P}$ are as follows: for a tuple $\bar{x}$ of distinct variables,

1. $\widehat{R}@p(\bar{x}, p)$:- $R@p(\bar{x})$ for each peer p in σ and each $R \in ext(p)$ (each peer can read its own extensional relations);

2. $\widehat{R}@p(\bar{x}, z) \text{ :- } acl@p(R, z), R@p(\bar{x})$ for each peer p in σ and each $R \in ext(p)$ (each peer z entitled to read $R@p$ can read all of its tuples);
3. for each rule $acl@p(Z, z) \text{ :- } R_1@p(\bar{x}_1), \cdots R_k@p(\bar{x}_k)$ in P_{pol},
   a rule $acl@p(Z, z) \text{ :- } \widehat{R}_1@p(\bar{x}_1, p), \cdots, \widehat{R}_k@p(\bar{x}_k, p)$;
4. for each rule $Z_0@z(\bar{x}_0) \text{ :- } R_1@p(\bar{x}_1), \cdots, R_k@p(\bar{x}_k)$ in P_{app} and for each intensional relation $R_0 \neq acl$ occurring in σ, a rule[1]
   $\widehat{R}_0@z(\bar{x}_0, y) \text{ :- } Z_0 = R_0,\ \widehat{R}_1@p(\bar{x}_1, y), \cdots, \widehat{R}_k@p(\bar{x}_k, y),\ \widehat{R}_1@p(\bar{x}_1, z), \cdots, \widehat{R}_k@p(\bar{x}_k, z)$
5. A rule $R@p(\bar{x}) \text{ :- } \widehat{R}@p(\bar{x}, p)$ for each $p \in dom(\sigma)$ and $R \in int(p)$ ($\widehat{R}@p$ defines the local facts visible at p).

The fourth item requires that both z (the next reader) and y (potential future readers) may access the facts in the body of the rule, in order be allowed to see the fact derived by the rule. The third item is the analog for *acl*. Note that (3.) is simpler than (4.) because the relation *acl* is only defined locally.

Clearly, the size of $\widehat{P}$ is linear in P and the image of σ. Moreover, it is independent of the data, i.e. $dom(\sigma)$ and I. Using $\widehat{P}$, we define two semantics for P: state semantics, and visibility semantics.

State semantics. State semantics provides for each peer the *local* intensional facts inferred by taking into account the combined effect of the access control rules and the application rules. More precisely, the state semantics of a d-datalog$_{ac}$ program P over schema σ is a mapping $[P]$ associating to each extensional instance I over σ the set of facts

$$[P](I) = \{R@p(\bar{a}) \in \widehat{P}(I) \mid p \in dom(\sigma), R \in int(p)\}\,.$$

One can easily verify by induction that $[P](I) \subseteq P(I)$. (Recall that $P(I)$ is the access-control-free semantics). The inclusion may be strict because the derivation of a fact at a peer p may be blocked because p does not have access to some data.

Visibility semantics. This semantics captures more broadly the facts at *all* peers that a given peer is allowed to see. Indeed, in addition to their local state provided by $[P]$, peers also have permission to see facts residing at *other* peers. The facts that they are allowed to see are specified by the relations $\widehat{R}@q(-, p)$ defined by the program $\widehat{P}$. We say that such a fact is *visible* by a peer p. For each p, we denote by $[P]_p^{\mathcal{V}}$ the mapping associating to each instance I over $ext(\sigma)$ the set of facts $\{R@q(\bar{u}) \mid \widehat{R}@q(\bar{u}, p) \in \widehat{P}(I)\}$. We refer to $[P]_p^{\mathcal{V}}$ as the *visibility semantics* for peer p. Clearly, for each p, $[P]_p^{\mathcal{V}}(I)$ and $[P](I)$ agree on $int(p)$.

Intuitively, if a fact $R@q(\bar{a})$ is visible by p, then p can access it by querying the relation $R@q$. More precisely, let P' be the program obtained by adding to P a rule $temp@p(\bar{u}) \text{ :- } R@q(\bar{u})$ for some new relation $temp@p$ and vector $\bar{u}$ of distinct variables. Then $temp@p(\bar{a}) \in [P'](I)$ iff $R@q(\bar{a}) \in [P]_p^{\mathcal{V}}(I)$, i.e. $R@q(\bar{a})$ is visible by p. Thus, visibility semantics can be reduced to state semantics by the addition of such rules.

In addition to state and visibility semantics, we consider in Section 4 the facts that a peer may *infer* from the visible ones, possibly circumventing the access control policy. We will refer to this as *implicit visibility.*

[1] Strictly speaking, equalities $Z = R_0$ are not allowed in d-datalog, but these can be easily simulated by substituting the variable by the constant everywhere in the rule.

Hiding access restrictions. The above access control mechanism may be too constraining in some situations. We next consider means of relaxing it. To do so, we introduce a *hide* annotation that can be attached to atoms in rule bodies, e.g., [hide $R@q(\bar{x})$]. Intuitively, such an annotation lifts access restrictions on $R@q(\bar{x})$ by "hiding its provenance".

We illustrate this feature with an example.

▶ **Example 6.** Consider the two rules:

- $\mathit{Album}@z(x)$:– $\mathit{Album}@\mathit{Bob}(x), \mathit{friend}@\mathit{Bob}(z)$
- $\mathit{Album}@z(x)$:– $\mathit{Album}@\mathit{Bob}(x)$, [hide $\mathit{friend}@\mathit{Bob}(z)$]

The first rule is used by Bob to publish his photos in all of his friends albums. Suppose Sue is a friend. Will the photos in $\mathit{Album}@\mathit{Bob}$ be transferred to $\mathit{Album}@\mathit{Sue}$? Yes, but only if Sue has read privileges on both $\mathit{Album}@\mathit{Bob}$ and $friends@Bob$. However, it may be the case that Bob wishes to keep his list of friends private, but still let his friends see his album pictures. He can do this by "hiding" the access restrictions on $friends@Bob$ as in the second rule. Intuitively, Bob is in effect reducing the protection level of the *friend* relation, in some sense "declassifying" it.

In the example, Bob declassifies *his own* extensional relation. As we will see, "hide" also allows a peer to declassify information received from *other* peers, thus overriding their access control restrictions. In the actual Webdamlog system [19], doing so requires the peer to have GRANT privilege on that piece of information. As previously mentioned, for simplicity we do not consider explicitly the GRANT mechanism here.

For further illustration, we show how the hide mechanism can be used to simulate accessing a relation with binding patterns [24].

▶ **Example 7.** Suppose that peer p wishes to export an extensional binary relation R with binding pattern bf. The intuition is that one cannot obtain the entire relation, but if one provides bindings for the first column, peer p will provide the corresponding values in the second column. This is done as follows:

- $\mathit{Seed}@p(x)$:– $S@q(x)$
- $Q@q(x, y)$:– $\mathit{Seed}@p(x)$, [hide $R@p(x, y)$]

Suppose the access control policy is such that p has read privilege on $S@q$, but q has no read privilege on $R@p$. Observe that $\mathit{Seed}@p$ is a copy of $S@q$, and $Q@q$ is the join of $\mathit{Seed}@p$ and $R@p$. Peer q cannot see $R@p$. But if q provides some values for the first column of $R@p$ (in relation $S@q$), then q will obtain in $Q@q$ the corresponding values for the second column of $R@p$.

Programs with *hide* are defined as follows.

▶ **Definition 8.** A *d-datalog$_{ac}$ program with hide* (denoted h-d-datalog$_{ac}$) over some schema σ consists of: (i) a d-datalog$_{ac}$ program $P = P_{app} \cup P_{pol}$; and (ii) a function h (called the *hide* function) whose domain h is the set P_{app} of rules[2], such that for each rule r, $h(r)$ is a strict subset of the atoms in the body of r. The pair (P_{pol}, h) forms the *policy* of the program.

As in Example 6, the function h is represented using annotations. More precisely, in each rule, the atoms in $h(r)$ are annotated with the keyword *hide*. For instance, the rule r that is A :– $B_1, \ldots B_5$ with $h(r) = \{B_2, B_4\}$ is denoted: A :– B_1, [hide B_2], B_3, [hide B_4], B_5.

We next consider how hide annotations modify the semantics of access control. The semantics for h-d-datalog$_{ac}$ programs is obtained by replacing item (4) of Definition 5 with:

[2] Because of the way we define access control rules, *hide* annotations would have no effect on them.

4'. for each application rule $Z_0@z(\bar{x}_0)$:– $R_1@p(\bar{x}_1), \cdots, R_k@p(\bar{x}_k)$ of P_{app}, for each intensional relation $R_0 \neq acl$ occurring in σ, and some new variable y, the rule $\widehat{R}_0@z(\bar{x}_0, y)$:– $Z_0 = R_0$, $\widehat{R}_1@p(\bar{x}_1, y_1), \cdots \widehat{R}_k@p(\bar{x}_k, y_k)$, $\widehat{R}_1@p(\bar{x}_1, q_1), \cdots \widehat{R}_k@p(\bar{x}_k, q_k)$ where for each i, if $R_i@p(\bar{x}_i)$ is not hidden in the rule, $y_i = y$ and $q_i = z$; and if it is hidden, $y_i = q_i = p$.

Note that this imposes that both y (a potential future reader) and z (the site that will host the fact) can read the facts in the body of the rule *that are not annotated by hide*, in order for the reader to be allowed to see the fact derived by the rule. For a h-d-datalog$_{ac}$ program P, we denote by $[P]$ the state semantics of P as defined by the above program.

The next result, namely Proposition 10, shows that the use of hide extends the expressive power of d-datalog$_{ac}$ relative to state semantics. (One can obtain a similar result for visibility semantics.) This is illustrated by the following example.

▶ **Example 9.** Consider a peer p that has a binary extensional relation $R@p$. Suppose we wish to specify that peer q sees from $R@p$ exactly the tuples of the form $(x, 0)$, and no other peer sees anything from $R@p$. As a first attempt, one might use an intensional relation R_{export} and the rule: $R_{export}@q(x, 0)$:- $R@p(x, 0)$.

However, either $acl@p(R, q)$ holds, so $R@p$ is entirely visible to q; or not, and $R_{export}@q$ is empty. Considering *hide*, assume the existence of some extensional fact $ok_q@p()$ that only q can read. Then there is a solution: $R_{export}@q(x, 0)$:– $ok_q@p()$, [hide $R@p(x, 0)$].

▶ **Proposition 10.** *There is a h-d-datalog$_{ac}$ program P over schema σ for which there is no d-datalog$_{ac}$ program $\bar{P}$ such that, for every extensional instance I over σ, $[P](I) = [\bar{P}](I)$.*

Thus, the *hide* construct strictly increases the expressivity of the language. In fact, we will show in Section 5 that h-d-datalog$_{ac}$ is in some sense expressively complete.

The complexity of access control. We consider throughout the paper the complexity of various problems related to access control. Typically, three kinds of complexity are considered in databases: data, query, and combined complexity. In d-datalog$_{ac}$, the distinction between data and schema/program is less clear. For instance, the set of peers affects both the schema and the data. If there are many peers, the global program may be large, even if each peer has a small program. To capture this situation, we consider a measure assuming that the size of the program *at each peer* is bounded. This gives rise to a novel notion of complexity that we call *locally-bounded combined complexity.* More precisely, for a decision problem whose input is an extensional instance I and a d-datalog$_{ac}$ program P over some schema σ:

- The *combined complexity* is computed as a function of $|I|$, $|P|$, and σ.
- The *data complexity* is computed as a function of $|I|$ only (σ and P are fixed).
- The *locally-bounded combined complexity* is computed as a function of $|I|$ and $|dom(\sigma)|$, assuming some fixed bound on the size of the program at each peer (so $|P|$ is linear in the number of peers).

We begin by establishing the complexity of checking the visibility of a fact.

▶ **Theorem 11.** *Let σ be a schema, I an extensional instance, and P a h-d-datalog$_{ac}$ program over σ. Determining whether a fact is in $[P]_p^{\mathcal{V}}(I)$ for some peer p has* PTIME*-complete data and locally-bounded combined complexity, and* EXPTIME*-complete combined complexity,*

While the data and the locally-bounded combined complexities are the same in this case, we will see later that the two differ in other settings, allowing to draw finer distinctions than the classical notions.

Static analysis of policies. To conclude this section, we briefly discuss the issue of *comparing* policies relative to a given application program, based on the visible facts they allow. This leads to the notion of a policy being *more relaxed than* another. By reduction from containment of datalog programs, one can show that this is undecidable for given policies and application program. As for datalog containment, one can consider restrictions for which the policy comparison can be performed, e.g., "frontier-guarded" rules [8]. As an alternative to comparing policies, one can consider applying syntactic transformations to a given policy in order to relax or tighten it. For example, augmenting the hide function of a program, or adding rules to P_{pol}, always results in a more relaxed policy. Due to space limitations, we do not further consider these issues here.

4 Implicit visibility

The purpose of access control is to analyse the ability of peers to see unauthorized information. As discussed in Section 3, a peer can access information by examining its own state or by querying relations of other peers. But can a peer infer more information beyond what is allowed according to the policy? We capture this using the notion of *implicit visibility* (i-visibility) that we formalize next. For this, we use the auxiliary notion of "visibility instance". For a program P over σ and a peer p, we say that an instance I_p over σ is a *visibility instance* of p if there is some instance J over $ext(\sigma)$ for which $I_p = [P]_p^{\mathcal{V}}(J)$. Now we define:

▶ **Definition 12.** Let P be a d-datalog$_{ac}$ program over some schema σ, p a peer and I_p a visibility instance for p. A fact $R@q(\bar{u})$ (for some q, R) is *i(mplicitly)-visible at* p given I_p, if for each instance J over $ext(\sigma)$ such that $[P]_p^{\mathcal{V}}(J) = I_p$, $R@q(\bar{u}) \in J \cup [P](J)$.

It turns out that facts beyond $[P]_p^{\mathcal{V}}(J)$ may be i-visible at peer p. To see how such information "leakage" can occur, suppose that we have a rule $acl@q(R,p) \coloneq Q@q(p)$, where $Q@q$ is an extensional relation. If peer p sees some fact in $R@q$, it can infer that it has access to $R@q$, so that $Q@q(p)$ holds, although the policy may not allow p to see $Q@q$. This may in turn provide additional information on other relations. Before exploring this formally, we introduce some restrictions of policies.

▶ **Definition 13.** Let σ be a schema and $P = P_{pol} \cup P_{app}$ a d-datalog program.

- The policy of P is *static* iff for each rule of P_{pol}, its body is empty;
- The policy of P is *simple* iff for each rule of P_{pol}, the atoms in its body are extensional;
- The policy of P is *local* for P_{app} iff for each peer p and rule of P_{pol} at p, the atoms in its body are either extensional, or intensional but not depending on non-local relations.

We can show that with static policy, no leakage can occur.

▶ **Proposition 14.** *Let P be a d-datalog$_{ac}$ program over σ with static policy. For each peer p and instance I over $ext(\sigma)$, the set of i-visible facts at p is precisely $[P]_p^{\mathcal{V}}(I)$.*

In contrast to the above, when P_{pol} contains arbitrary rules, i-visibility provides additional information, and is in fact undecidable.

▶ **Theorem 15.** *It is undecidable, given a d-datalog$_{ac}$ program P over σ, a visibility instance I_p for p, and a fact $R@q(\bar{u})$, whether $R@q(\bar{u})$ is i-visible at p given I_p. Moreover, undecidability holds even for programs with local access policies.*

The above undecidability result uses the fact that the *acl* relations are defined by datalog programs. We next show that i-visibility becomes decidable if recursion is disallowed in the definition of *acl* relations. The problem can be reduced to computing certain answers to datalog queries using exact UCQ views, which is known to be in co-NP [4]. However, using the fact that the views we use are particular UCQs, we can show that the complexity goes down to PTIME.

▶ **Theorem 16.** *The i-visibility problem for d-datalog$_{ac}$ programs with simple policies is decidable in* PTIME *(data complexity).*

The i-visibility problem with hide. We now turn to the problem of i-visibility for d-datalog$_{ac}$ programs with hide. The notions of visibility and i-visibility are adapted to this setting in the natural way. We first illustrate the fact that hide can lead to non-trivial i-visibility of facts, even when the acl policy is static.

▶ **Example 17.** Consider the following h-d-datalog$_{ac}$ program P where P_{pol} consists of the rule $acl@q(Q,p)$:- and P_{app} of the rules:

- $R_1@p(X)$:– $Q@q(), [\text{ hide } R@q(X,Y)]$;
- $R_2@p(Y)$:– $Q@q(), [\text{ hide } R@q(X,Y)]$.

Consider the p-visibility instance $\{R_1@q(a), R_2@q(b)\}$. Note that p does not have access to $R@q$. However, it is clear that $R@q(a,b)$ is i-visible at p.

The following result shows that i-visibility is undecidable for h-d-datalog$_{ac}$ programs even for static policies (when, by Proposition 14, no leakage occurs in the absence of *hide*). The proof is by reduction from finding certain answers to identity queries using exact datalog views, known to be undecidable [4].

▶ **Theorem 18.** *It is undecidable, given a h-d-datalog$_{ac}$ program P over σ with static policy, in which* hide *is applied only to extensional relations, a peer p, a p-visibility instance I_p, and an extensional fact $R@q(\bar{a})$, whether $R@q(\bar{a})$ is i-visible at p given I_p.*

Testing information leakage. The previous result concerned i-visibility for a given instance. We finally consider the problem of testing whether a d-datalog$_{ac}$ program has information leakage beyond that provided by the access control policy for *some* instance (the static analysis analog).

▶ **Definition 19.** A d-datalog$_{ac}$ program P *leaks information* at p if for some p-visibility instance I_p there exists some fact $R@q(\bar{a}) \notin I_p$ that is i-visible at p given I_p.

We show that one cannot generally decide whether a program leaks information. However, one can do so for programs with *simple* policies. The undecidability is proved using a reduction from datalog program containment. The 2EXPTIME algorithm for simple policies is by reduction to an exponential set of inclusions of datalog programs into UCQs.

▶ **Theorem 20.**

1. *It is undecidable, given a d-datalog$_{ac}$ program P and a peer p, whether P leaks information at p.*
2. *The problem is* 2EXPTIME*-complete if P has a simple acl policy.*

5 Achieving dissemination goals

We next consider the problem of achieving a specific data dissemination goal among peers, when a particular access control policy is imposed. The goal is specified by a d-datalog program. Clearly, a given goal may violate the policy, so it may be impossible to achieve it. We study the problem of determining whether achieving a goal is possible, and if not, how one might maximize what *can* be achieved. We then consider the issue of relaxing the access control policy in order to achieve the goal, using the *hide* mechanism. Not surprisingly, it is always possible to achieve a goal using *hide*. More interestingly, we will show how to do so while minimizing its use. But first, we consider what can be done without *hide*.

Strict adherence to the policy. Consider a policy P_{pol} and a goal d-datalog program P. We wish to know whether there is a d-datalog program P_{app} such that (i) P_{app} uses the relations of P and possibly additional intensional relations, and (ii) for each extensional instance I, $[(P_{pol} \cup P_{app})](I)$ and $P(I)$ agree on the intensional relations of P. In this case, we say that P_{app} *simulates* P under policy P_{pol}. We will see that it is generally impossible to find such a P_{app} without *hide*, and present restrictions on the policies that make it possible. When such a simulation does not exist, we will attempt to find a program that is as close as possible to the goal.

The next example illustrates how a policy may prevent achieving a goal even in the simplest setting. The example is more complicated than needed because we will also use it to illustrate finding a "maximum" simulation.

▶ **Example 21.** Consider the following policy and goal program:

$$\begin{array}{llllll} P_{pol} & acl@p(R_1,r) :- & ; & P & R@q(x) :- R_1@p(x); \\ & acl@p(R_2,r) :- & ; & & R@q(x) :- R_2@p(x); \\ & acl@p(R_1,q) :- & ; & & R@r(x) :- R@q(x) \end{array}$$

The d-datalog P does not simulate P under P_{pol} because q is not allowed to see the relation $R_2@p$ and therefore the relation $R@q$ does not hold tuples from $R_2@p$ under the policy P_{pol}. In such cases, we can try to find a program that is, in some sense, maximally achieves the goal. This is a nontrivial issue. In this example, a maximum application program is:

$$P_{app} : \quad R@q(x) :- R_1@p(x); \;\Big|\; R@r(x) :- R@q(x); \;\Big|\; R@r(x) :- R_2@p(x).$$

Note that $[(P_{pol} \cup P_{app})] \subseteq P$ but $[(P_{pol} \cup P)] \subset [(P_{pol} \cup P_{app})]$ (as mappings).

The first result states that one cannot decide whether a program can be simulated under a particular policy.

▶ **Theorem 22.** *It is undecidable, given a policy P_{pol} and a goal d-datalog program P, whether there exists a d-datalog program P_{app} without* hide *such that P_{app} simulates P under P_{pol}. This holds even if P_{pol} is static.*

If such a simulation is not possible, can we find a "maximum simulation"? Let P be a d-datalog program over some schema σ and P_{pol} a policy program over σ. A d-datalog program P_{app} without *hide* is a *maximum simulation* of P under P_{pol} iff

1. $[(P_{pol} \cup P_{app})] \subseteq P$, and
2. for each P'_{app} such that $[(P_{pol} \cup P'_{app})] \subseteq P$, $[(P_{pol} \cup P'_{app})] \subseteq [(P_{pol} \cup P_{app})]$.

The question of whether a maximum simulation always exists remains open. Moreover, there does not exist an algorithm building a maximum simulation, *if such exists*.

▶ **Theorem 23.** *There is no algorithm that computes, given a d-datalog program P and a policy P_{pol}, a maximum simulation without* hide *P_{app} of P under P_{pol}, whenever such a maximum simulation exists. This holds even for local policies.*

While it is not known whether a maximum simulation always exists, we present informally a plausible candidate for a maximum simulation of P under P_{pol} and explore its potential. The program, denoted by $\textsc{mac}(P_{pol}, P)$, is based on a simple idea: each peer collects all the extensional tuples that peer is allowed to see under P_{pol}, and then simulates P locally.

▶ **Definition 24.** Let P be a d-datalog program over some schema σ and P_{pol} a policy over the relations in σ. The program $P_{app} = \textsc{mac}(P_{pol}, P)$ is constructed as follows:

1. For all peers p, q, $p \neq q$ and each (extensional or intensional) relation $R@q$, P_{app} has an intensional relation $R\_q@p$ of the same arity as $R@q$. These relations allow p to perform a simulation of P with the data that p has access to.
2. For all peers p, q, $p \neq q$, and each extensional relation $R@q$, P_{app} has rules copying $R@q$ into $R\_q@p$, if $acl@q(R, p)$ holds.
3. Finally, for each peer p, P_{app} has rules that simulate P locally with the data that p has access to.

Observe how $\textsc{mac}(P_{pol}, P)$ interacts with P_{pol}. During the computation, some peer p may use rules in P_{pol} to derive a new fact $acl@p(R, q)$. This results in copying $R@p$ into $R\_p@q$ which may lead to the derivations of more facts at p.

Note the connection between $\textsc{mac}(P_{pol}, P)$ and P itself. By definition, $[(P_{pol} \cup P)] \subseteq [(P_{pol} \cup \textsc{mac}(P, P_{pol}))]$. However, the inclusion may be strict. For instance, P may try to transfer a fact from p to q via a peer r that is not allowed to see this fact whereas it is possible to send this fact directly (with a different rule) without violating access rights.

It turns out, surprisingly, that $\textsc{mac}(P_{pol}, P)$ is not always a maximum simulation of P under P_{pol}, and it is in fact undecidable whether $\textsc{mac}(P_{pol}, P)$ is a maximum simulation for some given (P_{pol} and P, even for local policies. However, $\textsc{mac}(P_{pol}, P)$ is a maximum simulation if P_{pol} is static.

▶ **Theorem 25.** *Let P_{pol} be a local policy and P a d-datalog goal program over σ.* (i) *It is undecidable whether the program $\textsc{mac}(P, P_{pol})$ is a maximum simulation of P under P_{pol}.* (ii) *If P_{pol} is static, then $\textsc{mac}(P, P_{pol})$ is a maximum simulation of P under P_{pol}.*

Besides ensuring the existence of a maximal simulation, a simple policy is of interest for another reason: it guarantees that, if there exists some application program simulating P under P_{pol}, then P itself simulates P under that policy (details omitted).

Declassifying information. Let us now consider the issue of achieving a goal at the cost of declassifying information, in other words using the *hide* construct. There is an immediate solution that would consist in modifying every rule of the goal program P by hiding the entire body. The goal would be satisfied, but in a brutal way: each derived fact would be visible to all peers.

It is possible to realize the goal in a much more controlled way as illustrated by Example 9. In that exemple, special relations of the form $ok_q@p$ are used to limit as much as possible the visibility of data. The example suggests the following mild technical assumptions: (†) for all distinct peers $p, q \in dom(\sigma)$, (1) σ contains a 0-ary extensional relation $ok_q@p$, and (2) extensional instances of σ are assumed to contain the fact $ok_q@p()$.

We next show that (†) is sufficient to guarantee that the *hide* construct allows achieving any goal program by declassifying no more information than necessary.

▶ **Theorem 26.** *Let σ satisfy (†.1). For each policy P_{pol} and a d-datalog goal program P over σ, there exists an application P_{app} with* hide *over the same σ such that, for each extensional instance I satisfying (†.2), P_{app} simulates P under P_{pol}; and on input I, a fact $R@p(u)$ is visible at $q \neq p$ for $(P_{pol} \cup P_{app})$ iff it is visible at q for $(P_{pol} \cup P)$.*

6 Accessing provenance

We considered so far the inference of individual facts using d-datalog$_{ac}$ rules, subject to an access control policy. In many applications, it is essential for inferred facts to be accompanied by *provenance* information. In this section, we extend our approach to access control to cover provenance. We adopt a simple model of provenance of a fact, consisting of derivation trees tracing the application of the rules at different peers that participated in the inference of the fact. To simplify the presentation, we ignore *hide*. The definition of provenance can be easily adapted to the presence of *hide* (a *hide* annotation in a rule results in truncating the corresponding portion of the proof tree) and the complexity results continue to hold.

Consider a d-datalog$_{ac}$ program P over schema σ. Let I be an extensional instance over σ, and $R@p(\bar{a})$ a fact in $P_{app}(I)$. A *provenance tree* for $R@p(\bar{a})$ is a derivation tree for $R@p(\bar{a})$ using P_{app} and I. Intuitively, we are interested in passing provenance information from peer to peer, so that a peer p not only knows that some fact $R@p(u)$ holds, but can also know how $R@p(u)$ has been derived.

▶ **Example 27.** Consider a schema σ with peers $\{p_0, p_1, p_2, p_3, p_4\}$, 0-ary extensional relations (propositions), $R@p_0$, $R@p_1$, and 0-ary intensional relations $S@p_2$, $S@p_3$, $S@p_4$. Let $I = \{R@p_0, R@p_1\}$. Consider the following application program:

$$P_{app} \quad S@p_2 \coloneq R@p_0; \;\Big|\; S@p_2 \coloneq R@p_1; \;\Big|\; S@p_3 \coloneq S@p_2; \;\Big|\; S@p_4 \coloneq S@p_3.$$

Note that $S@p_4 \in P_{app}(I)$ and has two provenance trees (linear in this case):

$$S@p_4 \leftarrow S@p_3 \leftarrow S@p_2 \leftarrow R@p_1 \qquad\qquad S@p_4 \leftarrow S@p_3 \leftarrow S@p_2 \leftarrow R@p_0$$

Suppose we have the following access control rules in addition to P_{app}:

$$P_{pol}: \quad acl@p_0(R, p_2) \coloneq ; \;\Big|\; acl@p_0(R, p_4) \coloneq ; \;\Big|\; acl@p_1(R, p_3) \coloneq ; \;\Big|\; acl@p_1(R, p_4) \coloneq .$$

Consider again the two provenance trees of $S@p_4 \in P_{app}(I)$. Neither satisfies the access control policy defined by P_{pol}. Indeed, the first tree violates the policy because p_2 does not have access to $R@p_1$. The second also violates the policy, because p_3 does not have access to $R@p_0$. If we add the access control rule: $acl@p_0(R, p_3) \;\coloneq$ then the second provenance tree satisfies the access control policy.

Note the difference between visibility of a fact A by a peer p and visibility of its *provenance*. In order for A to be visible by p, it suffices for each fact involved in its derivation to be visible by the corresponding intermediate peer, based on its own access permissions, independently derived. In other words, peers may justify their permissions by derivations independent of each other and of the actual derivation of A. Visibility of provenance imposes a stronger condition, as it requires each intermediate peer to have access to the *entire history* of the partial derivation of p. As seen in the example, a fact A may itself be visible by p but not have any provenance tree visible by p. More formally we have:

▶ **Definition 28** (Provenance access control). Let P be a d-datalog$_{ac}$ program over some schema σ and I an extensional instance over σ. A fact F *has visible provenance* if there exists a provenance tree T of F such that: For each internal node $R@p(\bar{a})$ in T and extensional fact $E@q(\bar{c})$ occurring in the subtree rooted at $R@p(\bar{a})$, we have that $acl@q(E, p) \in [P](I)$. For given P and I, $[P]^{prov}(I)$ denotes the set of facts that have visible provenance.

It is clear that visible provenance implies visibility. More precisely, one can show that for each P, σ, and each extensional instance I, $[P]^{prov}(I) \subseteq [P](I)$, but Example 27 shows the converse does not hold. We next show that, although the definition of provenance visibility is proof-theoretic, one can simulate it using a d-datalog program. However, unlike the program $\widehat{P}$ constructed earlier, the program simulating provenance visibility is exponential in the number of peers.

▶ **Proposition 29.** *Let P be a d-datalog$_{ac}$ program over some schema σ. There exists a d-datalog program (without access control) P^{prov} of size exponential in* $\mathrm{dom}(\sigma)$ *(and polynomial in σ and P if* $\mathrm{dom}(\sigma)$ *is fixed) with the same extensional relations as σ, such that for each extensional instance I, $[P]^{prov}(I)$ and $P^{prov}(I)$ agree on the intensional relations of σ.*

The program P^{prov} (in the proof of the previous result) uses constants to denote sets of peers. An alternative would consist in using an extension of d-datalog with nesting, in the style of extensions of datalog with nesting [6]. (Such a nested datalog is used in the implementation in [19].)

The d-datalog program P^{prov} is exponential in the set $dom(\sigma)$ of peers. Is it possible to avoid the exponential blowup? The following complexity result implies a negative answer (subject to usual assumptions). Consider the problem of deciding, given an extensional instance I and a program P, whether a fact is in $[P]^{prov}(I)$. Recall from Theorem 11 that the complexity of checking visibility of a fact has EXPTIME-complete combined complexity, and PTIME-complete data and locally-bounded combined complexity. Now we have:

▶ **Theorem 30.** *Let σ be a schema, I an extensional instance, and P a d-datalog$_{ac}$ program over σ. Determining whether a fact is in $[P]^{prov}(I)$ has* EXPTIME*-complete combined complexity,* PTIME*-complete data complexity and* PSPACE*-complete locally-bounded combined complexity.*

Theorems 11 and 30 show that provenance visibility has the same combined and data complexity as the standard semantics, but different locally-bounded combined complexity. As a corollary, the exponential blowup in Proposition 29 cannot be avoided (unless PTIME = PSPACE). This highlights the usefulness of this complexity measure in making finer distinctions than the classical ones.

7 Related work

Database security and access control have been studied in depth (e.g., see [10]) since the earliest works on System R [26] and Ingres [27].

Controlling access to intensional facts in deductive languages is related to managing virtual views in SQL, which is handled differently among various database systems. When an authorized user accesses a view, it is usually evaluated with the privileges of the defining user ("definer's rights"). Some systems (e.g. mySQL) allow the creator of a view to specify that later access to the view will be with respect to the privileges of the invoker of the view ("invoker's rights"). This is similar in spirit to our approach.

The access control model we have described is fine-grained, unlike the SQL standard. Lefevre et al [18] propose a fine-grained access control model for implementing personal privacy policies in a relational database. They use query modification to enforce their policies, as we do, but their policy model and implementation are oriented towards a centralized database system. A commercial example of fine-grained access control is Oracle's Virtual Private Database (VPD), which supports access control at the level of tuples or cells. VPD allows an administrator to associate an external function with a relation and automatically modifies queries to restrict access by tuple or cell. Alternative semantics for fine-grained access control have been investigated thoroughly [18, 25, 29]. Rizvi et al. [25] distinguish between Truman and Non-Truman models (the expression is motivated by the movie *The Truman Show* where the hero is unaware that he lives in an artificial environment). Query answers in our system follow the Truman paradigm: queries are not rejected because of lack of privilege but the user's privileges limit the answers that are returned.

Fine-grained access control is also studied in [13], where predicate-based specification of authorization is supported. The inference of sensitive data from exposed data (that we study here under the name of i-visibility) is related to a notion studied in [30].

Our model of access control shares some features with the model of reflective database access control (RDBAC) in which access policies can be specified in terms of data contained in any part of the database. Olson et al. [21] formalize RDBAC using a version of datalog with updates [11] but their model does not include distribution, delegation, or the use of provenance. In Cassandra [17], access rights are specified using a language based on datalog with constraints. The language supports complex specifications based on "user roles". On the other hand, fine-grained access control is not considered.

The use of provenance as a basis for access control was first noted in the context of provenance semirings [15, 7]. A security semiring can contain tuple-level security annotations and define the rules by which they are propagated to query results. Another example of provenance-based access control is the work of Park et al. [23] in which access decisions are based on a transactional form of provenance.

The emergence of social networks and other Web 2.0 applications has led to new forms of access control. In online social networks, the distinguishing feature is that access control policy is expressed in terms of network relationships amongst members [12, 14], and this is one of the motivations of the model we presented. However, the model is intended to support the diverse requirements of access control in a variety of distributed applications.

The Webdamlog language was first described in [3] as a version of distributed datalog in which peers exchange not only facts, but also rules. Expressiveness and semantic issues were formally investigated, but access control was not considered. As already mentioned, we build here on the Webdamlog access control mechanism of [19]. Its main novelty is the specification of the access rights on an inferred tuple based on the access rights on the tuples used to derive it. The full access control mechanism of [19] is richer than the one described here, notably using also GRANT and WRITE privileges. They present an open-source implementation (with Bud [9] inside), and an experimental evaluation showing that the computational cost of access control is modest. In the Webdam project context, cryptographic techniques for enforcing access control in a distributed manner (and detecting security violations) have been considered in [5]. The techniques proposed there can be combined with those presented here.

Security in distributed systems has primarily focused on issues of remote authentication, authorization, and protection of data and distributed trust; such issues are outside the scope of our present work [1, 22].

8 Conclusion

We presented a first formal study of provenance-based access control in distributed datalog inspired by the collaborative access control mechanism of [19]. The results highlight the subtle interplay between declarative distributed computation, coarse-grained and fine-grained access control. Starting from coarse-grained access control on local extensional relations, distributed datalog computation yields fine-grained access control on derived facts based on their provenance. We also considered access control on tuples equipped with explicit provenance. We briefly studied the problem of information leakage, occurring when peers can infer unauthorized information from authorized data. We established the complexity of access control, as well as of various analysis tasks, such as detecting information leakage, comparing access policies, or the ability to achieve specified goals under a given policy. A challenging aspect of the framework is the fluid boundary of schema, data, and program, that has an impact on both semantics and complexity. For example, this led us to define a new complexity measure, locally-bounded combined complexity, that can make more subtle distinctions than classical data and query complexity.

In this first investigation, we have ignored some important aspects of the Webdamlog system presented in [19]. In Webdamlog, "nonlocal rules" allow dynamic deployment of rules from one peer to another. Most of the results presented here extend to non-local rules. We also ignored here the GRANT and WRITE privileges of Webdamlog. These raise new subtle issues, notably when access control updates are considered. Finally, delegation in Webdamlog allows peers to assign tasks to other peers. The access control of delegation is supported in Webdamlog by a mechanism called "sandboxing" that also raises interesting issues. These are left for future research.

Acknowledgement. The authors gratefully acknowledge the support of the French ANR Aggreg project ANR-14-CE25-0017 (Pierre Bourhis) and of the U.S. National Science Foundation under award IIS-1422375 (Victor Vianu).

References

1 Martin Abadi, Michael Burrows, Butler Lampson, and Gordon Plotkin. A calculus for access control in distributed systems. In *ACM Trans. Program. Lang. Syst.*, pages 706–734, 1993.

2 Serge Abiteboul, Emilien Antoine, Gerome Miklau, Julia Stoyanovich, and Jules Testard. [Demo] rule-based application development using WebdamLog. In *SIGMOD*, 2013.

3 Serge Abiteboul, Meghyn Bienvenu, Alban Galland, and Emilien Antoine. A rule-based language for Web data management. In *PODS*, 2011.

4 Serge Abiteboul and Oliver M Duschka. Complexity of answering queries using materialized views. In *PODS*, pages 254–263. ACM, 1998.

5 Serge Abiteboul, Alban Galland, and Neoklis Polyzotis. A model for web information management with access control. In *WebDB Workshop*, 2011.

6 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases.* Addison-Wesley, 1995.

7 Y. Amsterdamer, D. Deutch, and V. Tannen. Provenance for aggregate queries. In *PODS*, 2011.

8 Vince Bárány, Balder ten Cate, and Martin Otto. Queries with guarded negation. *PVLDB*, 5(11):1328–1339, 2012.

9 Berkeley Orders Of Magnitude Project. Bloom programming language. URL: `http://www.bloom-lang.net/`.

10 Elisa Bertino and Ravi Sandhu. Database security-concepts, approaches, and challenges. *Dependable and Secure Computing, IEEE Transactions on*, 2(1):2–19, 2005.

11 Anthony Bonner. Transaction datalog: A compositional language for transaction programming. In *DBPL*. Springer, 1997.

12 Barbara Carminati, Elena Ferrari, Raymond Heatherly, Murat Kantarcioglu, and Bhavani Thuraisingham. A semantic web based framework for social network access control. In *SACMAT*, pages 177–186, 2009. `doi:10.1145/1542207.1542237`.

13 Surajit Chaudhuri, Tanmoy Dutta, and S Sudarshan. Fine grained authorization through predicated grants. In *ICDE*, pages 1174–1183. IEEE, 2007.

14 Elena Ferrari. *Access Control in Data Management Systems*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2010.

15 Todd J. Green, Grigoris Karvounarakis, and Val Tannen. Provenance semirings. In *PODS*, pages 31–40, 2007.

16 Guy Hulin. Parallel processing of recursive queries in distributed architectures. In *VLDB*, pages 87–96, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc.

17 Avinash Lakshman and Prashant Malik. Cassandra: A decentralized structured storage system. *SIGOPS Oper. Syst. Rev.*, 44(2):35–40, 2010.

18 Kristen LeFevre, Rakesh Agrawal, Vuk Ercegovac, Raghu Ramakrishnan, Yirong Xu, and David DeWitt. Limiting disclosure in hippocratic databases. In *VLDB*, pages 108–119. VLDB Endowment, 2004.

19 Vera Zaychik Moffit, Julia Stoyanovich, Serge Abiteboul, and Gerome Miklau. Collaborative access control in WebdamLog. In *SIGMOD*, 2015.

20 Wolfgang Nejdl, Stefano Ceri, and Gio Wiederhold. Evaluating recursive queries in distributed databases. *Knowledge and Data Engineering, IEEE Transactions on*, 5(1):104–121, 1993.

21 Lars E. Olson, Carl A. Gunter, and P. Madhusudan. A formal framework for reflective database access control policies. In *CCS'08: Proceedings of the 15th ACM conference on Computer and communications security*, pages 289–298, New York, NY, USA, 2008. ACM. `doi:10.1145/1455770.1455808`.

22 M. Tamer Özsu and Patrick Valduriez. *Principles of Distributed Database Systems, Third Edition*. Springer, 2011.

23 Jaehong Park, Dang Nguyen, and R. Sandhu. A provenance-based access control model. In *International Conference on Privacy, Security and Trust*, pages 137–144, 2012. `doi:10.1109/PST.2012.6297930`.

24 Anand Rajaraman, Yehoshua Sagiv, and Jeffrey D Ullman. Answering queries using templates with binding patterns. In *PODS*, pages 105–112. ACM, 1995.

25 Shariq Rizvi, Alberto Mendelzon, S. Sudarshan, and Prasan Roy. Extending query rewriting techniques for fine-grained access control. In *SIGMOD*, pages 551–562, New York, NY, USA, 2004. ACM Press. `doi:10.1145/1007568.1007631`.

26 Patricia G. Selinger, Morton M. Astrahan, Donald D. Chamberlin, Raymond A. Lorie, and Thomas G. Price. Access path selection in a relational database management system. In *SIGMOD Conference*, pages 23–34, 1979.

27 Michael Stonebraker, Gerald Held, Eugene Wong, and Peter Kreps. The design and implementation of INGRES. *ACM Trans. Database Syst.*, 1(3):189–222, September 1976. `doi:10.1145/320473.320476`.

28 Prasang Upadhyaya, Magdalena Balazinska, and Dan Suciu. Automatic enforcement of data use policies with datalawyer. In *SIGMOD*, pages 213–225, 2015.

29 Qihua Wang, Ting Yu, Ninghui Li, Jorge Lobo, Elisa Bertino, Keith Irwin, and Ji-Won Byun. On the correctness criteria of fine-grained access control in relational databases. In *VLDB*, pages 555–566, 2007.

30 Hong Zhu, Jie Shi, Yuanzhen Wang, and Yucai Feng. Controlling information leakage of fine-grained access model in dbmss. In *WAIM*, pages 583–590. IEEE, 2008.