Belief Reasoning in MLS Deductive Databases

Hasan M. Jamil
Department of Computer Science
Mississippi State University, USA
`jamil@CS.MsState.Edu`

Abstract

It is envisaged that the application of the multilevel security (MLS) scheme will enhance flexibility and effectiveness of authorization policies in shared enterprise databases and will replace cumbersome authorization enforcement practices through complicated view definitions on a per user basis. However, as advances in this area are being made and ideas crystallized, the concomitant weaknesses of the MLS databases are also surfacing. We insist that the critical problem with the current model is that the *belief* at a higher security level is cluttered with irrelevant or inconsistent data as no mechanism for attenuation is supported. Critics also argue that it is imperative for MLS database users to theorize about the belief of others, perhaps at different security levels, an apparatus that is currently missing and the absence of which is seriously felt.

The impetus for our current research is this need to provide an adequate framework for belief reasoning in MLS databases. We demonstrate that a prudent application of the concept of inheritance in a deductive database setting will help capture the notion of declarative belief and belief reasoning in MLS databases in an elegant way. To this end, we develop a function to compute belief in multiple modes which can be used to reason about the beliefs of other users. We strive to develop a poised and practical logical characterization of MLS databases for the first time based on the inherently difficult concept of non-monotonic inheritance. We present an extension of the acclaimed Datalog language, called the MultiLog, and show that Datalog is a special case of our language. We also suggest an implementation scheme for MultiLog as a front-end for CORAL.

Key Words: MLS databases, belief assertion, reasoning, inheritance and overriding, deductive databases.

1 Introduction

Research into multilevel secure (MLS) relational models has intensified in recent years as defense and corporate database applications demand more flexible and fine grain, yet, effective, authorization protocols for increased sharing of knowledge [8, 12, 19, 24, 27]. Traditional ways of defining fine grain authorization were to create complicated views on a per user basis that essentially limited access to an entire column in a relation in an *all* or *nothing* fashion. Thus authorization at the individual data level could not be defined easily.

Multilevel secure relational models have been around for some time and have attracted significant attention from established academician and researchers. Prototypes and commercial MLS databases are being built although a clear consensus on the gross features is yet to be reached. It turns out that the MLS model has very stringent and complicated security policies and capturing all these policies has proven to be very difficult. However, the abundant recent research into MLS relational databases shows that MLS policies are well suited for sensitive defense and corporate database applications in a natural way. Research has also established that the model is effective and outperforms current relational authorization principles for such applications. Generally, according to this view, users with a security clearance level c would be able to access only the data that are equal or lower in security classification than c. This view of data and users is in perfect agreement with the traditional view of corporate knowledge and administrators' control on such knowledge. However, the simplicity of the model has been found to be deceiving and the implementation of the apparently simple concept has demanded significant investigations and development.

Recent research shows that users in the MLS model potentially have a cluttered view and ambiguous belief of data [16]. The extraction process of knowledge and belief about data from such databases is manual and error prone. Ad hoc knowledge extraction is quite an undertaking, and understanding what others believe is not easily possible. Critics argue that it is imperative for users to theorize about the belief of other users at different levels. Current models, unfortunately, do not provide any support to this end. The aim of this research is to address some of these issues that have been identified as bottlenecks for contemporary proposals. In the following sections, we expose the limitations of the representative proposals using an example that we adapt from the literature. We suggest possible functional enhancements and strive to develop a logic based query language, called *MultiLog*, for MLS databases.

Instead of developing yet another language, in this paper we extend the acclaimed Datalog language syntactically, define an operational semantics, and then explain the functionality of MultiLog by rewriting it in a variant of Datalog - i.e., CORAL. We report that while we view the current proposal as an implementation framework for MultiLog, a more theoretical treatment of MultiLog is possible [14] that gives MultiLog the status of a query language in its own right.

1.1 Outline of the Paper

We have planned the presentation of the current research as follows. We first present a brief introduction to multilevel relational model in section 2. We refrain from presenting a detailed discussion on the MLS model for the sake of conciseness and in order to focus our attention to the development of a belief model and a declarative query language. Interested readers are referred to [12] and [16] for an eloquent discussion on MLS relational models and belief assertion in MLS models respectively. Then in section 3, we expose the limitations of well known and representative MLS proposals in the literature. We pinpoint the weaknesses of these proposals especially with respect to belief models of users. In this section, we also discuss a possible belief model on intuitive grounds and introduce the idea of a parametric belief function. Section 4 discusses contemporary research and identifies the contributions of MultiLog in the light of these proposals. We develop the syntax and semantics of MultiLog in section 5. The semantics is given by developing the notion of a simple and consistent database, and by giving an operational semantics of MultiLog for such databases. A reduction semantics is also presented in section 6 which serves as the implementation framework for our language. We prove that the two semantics are logically equivalent and consistent. We then discuss several implementation related issues in section 7, some of which remain part of our future investigation. We summarize and conclude in section 8.

2 The Multilevel Relational Data Model

In this section, we present a very brief introduction to the MLS relational model for the sake of completeness. Multilevel relational model is shaped after Bell-LaPadula security model. In this security model, data are called the *objects* and processes or users are called *subjects*. Every object is assigned a security classification, and every subject is assigned a clearance. The security classification of objects and clearances of subjects is collectively referred to as *access classes* (or *levels*). An access class has two components – a hierarchy of levels (e.g., Top Secret (T), Secret (S), Classified (C) and Unclassified (U)[1]), and an unordered set of categories (e.g., NATO, Army, Navy, etc.). Access classes are partially ordered in a lattice such that given two access classes c_1 and c_2, $c_1 \geq c_2$ if, and only if, the hierarchical component of c_1 is greater than or equal to that of c_2 and the categories in c_1 is a superset of those in c_2. In this case, we say that level c_1 *dominates* level c_2. For simplicity, we will drop the category components of access classes throughout this paper without the loss of any generality.

The restrictions imposed by Bell-LaPadula model may be summarized as follows:

1. A subject is allowed to have read access to an object if the subject has a clearance that dominates the classification of the object. This is called the *simple security property.*
2. A subject is allowed a write access to an object only if the subject's clearance is dominated by the object's classification. This is known as the *$\star$-property.*

[1]We may assume that level T is higher in security classification than level S, denoted T > S. We also assume that S > C, and finally C > U.

A discussion on the implications of these properties may be found in [12]. But it is important to mention here that these properties are necessary but not sufficient for effective security enforcement.

Bell-LaPadula restrictions imply that the subjects at different clearance levels see different versions of a multilevel relation. For example, a user with a clearance level c will see only those data that have classifications dominated by c. We will discuss the views at different levels later in the next section using an example.

Formally, a multilevel relation (similar to classical relations) consists of two parts: scheme and instances, defined as follows:

Definition 2.1 (Scheme) Let $A_1, \ldots, A_n$ be *data attribute* names, $C_1, \ldots, C_n$ be *classification attribute* names for each of the data attributes, and TC be the *tuple-class attribute*. Let the domains of each data attribute A_i be D_i. Let the domain of each C_i be specified by a range $[L_i, H_i]$ such that it defines a sub-lattice of access classes ranging from L_i up to H_i. let the domain of TC be the range $[lub^2\{L_i : i = 1, \ldots, n\}, lub\{H_i : i = 1, \ldots, n\}]$. Then $R(A_1, C_1, A_2, C_2, \ldots, A_n, C_n, TC)$ is a multilevel relation scheme.

The classification attributes (C_is) in a scheme S registers the security classification of the attribute values, while the tuple classification TC registers the access class c where the tuple was inserted/updated. In general, a user is allowed to see the entire tuple (including TC) if the user's clearance level dominates the tuple classification c.

Definition 2.2 (Instance) Let $R(A_1, C_1, A_2, C_2, \ldots, A_n, C_n, TC)$ be a multilevel relation scheme. Then, a multilevel relation instance r is a set of ordered tuples of the form $(a_1, c_1, a_2, c_2, \ldots, a_n, c_n, tc)$ such that each $a_i \in D_i$, or $a_i = \perp$, and $tc = lub\{c_i : i = 1, \ldots, n\}$. If $a_i \neq \perp$ then $c_i \in [L_i, H_i]$. Also, $c_i \neq \perp$ for any a_i.

Definition 2.3 (View at c) Let r be a multilevel relation instance over the scheme $R(A_1, C_1, A_2, C_2, \ldots, A_n, C_n, TC)$. Then, the view at access class c is a relation instance r_c derived from r such that r_c is a set of ordered tuples of the form $(a_1, c_1, a_2, c_2, \ldots, a_n, c_n, tc)$ where each $a_i \in D_i$, or $a_i = \perp$, $c \geq c_i$ and $tc = lub\{c_i : i = 1, \ldots, n\}$.

In a view at access class c, for every tuple $t \in r$, $t[A_i]$ is in r_c if, and only if $c \geq t[C_i]$, otherwise $t[A_i] = \perp$ in r_c, i.e., when $c \leq t[C_i]$.

Multilevel relations are required to satisfy several integrity properties. We will discuss these properties in relation to MultiLog in definitions 5.3 and 5.4. We, however, introduce the notion of *apparent primary keys* in this section. Since multilevel relations have different instances at different access classes, the notion of keys becomes clouded because a relation instance is now a collection of sets of tuples rather than a single set of tuples. The user specified *primary key* cannot be used as the primary key anymore. Consequently, the primary key is known as the *apparent primary key*, and is denoted by AK. The classification of AK is denoted by C_{AK}. It is easy to show that in multilevel relations, $AK, C_{AK}, C_i \rightarrow A_i$ holds for every data attribute A_i. As such, the primary key of a multilevel relation can be defined as $AK \cup C_{AK} \cup C_R$ where, C_R is the set of classification attributes for data attributes not in AK [12]. Figure 1 shows an example of a multilevel relation.

[2]Least upper bound.

3 Belief Models in MLS Databases

Consider the following example adapted from Jukic and Vrbsky [16]. This example exposes some of the difficulties faced by users in contemporary MLS databases trying to form a belief about the data. The apparent primary key [12], *Starship*, of the relation *Mission* is underlined. Note that the relation satisfies the functional dependency $AK, C_{AK}, C_i \rightarrow A_i$ for all attribute A_i in the scheme of *Mission*[3], where AK is the apparent key (i.e., *Starship*), C_{AK} is the security classification of the apparent key, and C_i is the security classification of the attribute A_i, and thus satisfy the *polyinstantiation integrity* [12]. Also notice that tuples t_4 and t_5 are possible through a series of updates if *required polyinstantiation* [12] is enforced.

Tid	**Starship**		**Objective**		**Destin**		**TC**
t_1	Avenger	S	Shipping	S	Pluto	S	S
t_2	Atlantis	U	Diplomacy	U	Vulcan	U	S
t_3	Voyager	U	Spying	S	Mars	U	S
t_4	Phantom	U	Spying	S	Omega	U	S
t_5	Phantom	C	Supply	S	Venus	S	S
t_6	Atlantis	U	Diplomacy	U	Vulcan	U	C
t_7	Atlantis	U	Diplomacy	U	Vulcan	U	U
t_8	Voyager	U	Training	U	Mars	U	U
t_9	Falcon	U	Piracy	U	Venus	U	U
t_{10}	Eagle	U	Patrolling	U	Degoba	U	U

Figure 1: MLS relation *Mission*(Startship, C_1, Objective, C_2, Destination, C_3, TC).

The interpretation of the above relation is obviously less than simple and there exist many opinions. This is partly because the interpretation is mostly application or user dependent. Jajodia and Sandhu [12] define interpretation at a given level in terms of visibility rules and classify visible tuples as *true*, or *cover stories*. The following query in the framework of [12] would produce the entire *Mission* relation when submitted by an user with a *S* level clearance. It will, however, produce the relation in figure 2 if submitted by a *U* level user[4].

```
select *
from mission
```

Tid	**Starship**		**Objective**		**Destin**		**TC**
t_4	Phantom	U	$\perp$	U	Omega	U	U
t_7^*	Atlantis	U	Diplomacy	U	Vulcan	U	U
t_8^*	Voyager	U	Training	U	Mars	U	U
t_9	Falcon	U	Piracy	U	Venus	U	U
t_{10}	Eagle	U	Patrolling	U	Degoba	U	U

Figure 2: *U* level view of *Mission*.

In contrast to the above, a *C* level user's view is the relation in figure 3. We point out here that the tuples t_4 and t_5 do not *subsume* each other as discussed in [12]. Subsumption helps clear the unwanted and irrelevant tuples while hiding the existence of higher level tuples. Ordinarily a *null* value in a tuple will show up only if part of a lower level tuple is updated by a higher level user who possibly left the key classification unchanged. This will force polyinstantiating the database to hide the higher level update from the lower level users. However, the lower key classification that remained as part of the higher level tuple will now force introduction of null values as the key remained visible to a lower level user. Fortunately, the nulls will be subsumed (in most cases) by the lower level tuple with non null values. But if the lower level tuple is now deleted, and the key classification of the higher level tuple stays unchanged, a tuple with null values will surface as it did in the case of the tuples t_4 and t_5.

Tid	**Starship**		**Objective**		**Destin**		**TC**
t_4	Phantom	U	$\perp$	U	Omega	U	C
t_5	Phantom	C	$\perp$	C	$\perp$	C	C
t_6^*	Atlantis	U	Diplomacy	U	Vulcan	U	C
t_8^*	Voyager	U	Training	U	Mars	U	U
t_9	Falcon	U	Piracy	U	Venus	U	U
t_{10}	Eagle	U	Patrolling	U	Degoba	U	U

Figure 3: A *C* level user view of the *Mission*.

It is our contention that such tuples compromise the security of the MLS databases, perhaps due to unawareness or due to intentional malice on the part of the higher level user. The point here is that current models do not prevent this from happening. In this instance, the *C* level user knows that a cover story has been given to the *U* level user but fails to determine the cover story. Furthermore, she now knows that she was also given a cover story by a higher level user. To our knowledge, this phenomenon was not been discovered in any earlier research. We call tuples such as t_4 and t_5, *surprise stories.*

However, it is easy to observe that forming an opinion about the visible data remains the responsibility of the user. Users proceed to determine the meaning of tuples by making extensive comparisons with other tuples. Only after they perform this extra step can they know whether the tuples are cover stories or real tuples. We maintain that it is still unclear as to what to make of the lower level true tuples or the tuples with null values that flow from higher levels, the surprise stories. Should a user believe such tuples or ignore them? Is it really necessary to assume that just because a tuple was contributed by a lower level user it is useless, independent of the existence of a higher level tuple that possibly contradicts the lower level tuple? There has been no simple answer to these questions.

Tid	**Starship**		**Objective**		**Destin**		**TC**
t_1	Avenger	S	Shipping	S	Pluto	S	S
t_2	Atlantis	UCS	Diplomacy	UCS	Vulcan	UCS	UCS
t_3	Voyager	US	Spying	S	Mars	US	S
t_4	Phantom	US	Spying	U-S	Omega	US	U-S
t_4'	Phantom	US	Spying	S	Omega	US	S
t_5	Phantom	CS	Supply	S	Venus	S	S
t_5'	Phantom	CS	Supply	C-S	Venus	C-S	C-S
t_8	Voyager	US	Training	U-S	Mars	US	U-S
t_9	Falcon	U-S	Piracy	U-S	Venus	U-S	U-S
t_{10}	Eagle	U	Patrolling	U	Degoba	U	U

Figure 4: Jukic and Vrbsky's view of *Mission*

Jukic and Vrbsky [16] addressed this issue of belief formation in [16]. They, however, use a richer set of security labels

[3]Note that *Tid* is not part of the scheme. We use it for the convenience of reference.

[4]The tuples identified with asterisk *subsumes* [12] other tuples in the view.

in which they encode the visibility rules and decide the status of a tuple at a given level. They would represent the *Mission* relation as shown in figure 4.

The interpretation assigned to each tuple in *Mission* in their framework is shown in figure 5. We consider this interpretation to represent somewhat of a departure from Jajodia and Sandhu, but it actually provides a framework for asserting beliefs of the users directly.

Tid	**U level**	**C level**	**S level**
t_1	invisible	invisible	true
t_2	true	true	true
t_3	invisible	invisible	true
t_4	true	irrelevant	*cover story*
t'_4	invisible	invisible	true
t_5	invisible	invisible	true
t'_5	invisible	true	cover story
t_8	true	irrelevant	cover story
t_9	true	irrelevant	*mirage*
t_{10}	true	irrelevant	irrelevant

Figure 5: Interpretation of tuples at different levels

3.1 Dynamic Belief Reasoning

Our contention is that both these models of belief are inadequate and somewhat stringent. The Jajodia-Sandhu model is too basic where users are left to discover the truth. On the other hand, Jukic-Vrbsky model is too restrictive and has only fixed interpretations. Users in these frameworks really do not have any reasoning capabilities as the interpretations are already given. We believe a middle ground is warranted where the user is given the choice to reason and theorize about the beliefs of others and decide how she wants to believe information visible to her.

In this direction, we assert that users should be given linguistic tools to view data as well as to construct meaning of the visible data. For example, the user may take a *firm* view of the data and insist that whatever is created at her security level *only* are correct and believable data. Thus lower level data are of no value. For example, a firm C level view of *Mission* relation could be as shown in figure 6.

Tid	**Starship**		**Objective**		**Destin**		**TC**
t_6	Atlantis	U	Diplomacy	U	Vulcan	U	C

Figure 6: Conservative or firm view of *Mission* at level C.

On the other hand, one may want to believe the best she can in the absence of any information at her own level, either in a monotonic way or in an overriding fashion. The monotonic version of the best possibility can be called an *optimistic* view. In this view, an user accumulates all possible data that are visible and considers important and thus believes the data. An optimistic view of *Mission* relation is shown in 7 for a C level user. Contrast this view with the C level user view in Jajodia and Sandhu shown in figure 3. In the optimistic view, the TC values become C while in figure 3, it retains the original source level information.

The overriding version of the best possibility is called the *cautious* view. In this view, the *visible* information at a given

Tid	**Starship**		**Objective**		**Destin**		**TC**
t_4	Phantom	U	$\perp$	U	Omega	U	C
t_5	Phantom	C	$\perp$	C	$\perp$	C	C
t_6	Atlantis	U	Diplomacy	U	Vulcan	U	C
t_8	Voyager	U	Training	U	Mars	U	C
t_9	Falcon	U	Piracy	U	Venus	U	C
t_{10}	Eagle	U	Patrolling	U	Degoba	U	C

Figure 7: An optimistic view of *Mission* at level C.

level that has the highest security classification is retained and others filtered out. The fundamental assumption under this view is that a higher level information is more reliable and the lower level counterpart is a cover story. The table in figure 8 presents a cautious view at level C. It is interesting to note here that if the security levels form a partial order, and not a total order, a cautious view may still have conflicting information due to multiple incomparable sources (levels). This is reminiscent of the problem in object oriented systems with multiple inheritance. Consequently, we must settle for multiple models and associated unpredictability.

The process of computing the cautious view presented in figure 8 from the C level view in figure 3 deserves some additional explanations. Note that tuple t_4 does not subsume t_5 and vice versa. In the cautious view, for every pair of tuples u and v such that $u[AK] = v[AK]$, we create a tuple t such that for every attribute $A_i \in R$, $t[A_i] = u[A_i], t[C_i] = u[C_i]$ if $u[C_i] \geq v[C_i]$, otherwise $t[A_i] = v[A_i], t[C_i] = v[C_i]$. Notice that the process of creating t is reminiscent of inheritance with overriding in inheritance systems. Here, if a level dominates another level, the values at the dominating level overrides the values at the lower levels[5]. Hence, in figure 8, we have tuple t_5, while t_4 is missing.

Tid	**Starship**		**Objective**		**Destin**		**TC**
t_5	Phantom	C	$\perp$	C	$\perp$	C	C
t_6	Atlantis	U	Diplomacy	U	Vulcan	U	C
t_8	Voyager	U	Training	U	Mars	U	C
t_9	Falcon	U	Piracy	U	Venus	U	C
t_{10}	Eagle	U	Patrolling	U	Degoba	U	C

Figure 8: Cautious view of *Mission* at level C.

While we have discussed only three possible views of MLS data in the foregoing presentation, we recognize the fact that other views of the MLS relations are conceivable. In fact, Cuppens [7] proposes several such views, and we trust that our views subsume all the views he has proposed, namely the additive view, the suspicious view and the trusted view.

3.2 A Parametric Belief Function

Consider the following query:

List all starships that are spying on Mars without any doubt.

[5]Put it in object-oriented terminologies, lower access classes/classifications are treated like superclasses, and the higher access classes/classifications are treated like subclasses.

A possible extended SQL query[6] is shown below. This is assuming that the visibility in all possible ways leads to a belief without doubt[7]. Note specially the simplicity of the SQL query below. An equivalent version of the same query using the syntax in either [12] or [27] would be far more complicated as they do not support *belief modes.*

```
user context u
select starship
from mission m
where m.starship in (select starship
                from mission
                where destination = mars and objective = spying
                believed cautiously)
        intersect
                (select starship
                from mission
                where destination = mars and objective = spying
                believed firmly)
        intersect
                (select starship
                from mission
                where destination = mars and objective = spying
                believed optimistically)
```

The preceding discussion demonstrates that a linguistic instrument to compute ad hoc beliefs in multiple modes adds to the strength of the language. Ad hoc belief computation helps reasoning with the beliefs of users and facilitates understanding the knowledge base better. It also increases the expressibility of the query language. In this section, we introduce a parametric belief function for MLS databases as a candidate for belief computation model.

The belief function discussed below assumes that the security labels form a partial order and that the set of belief modes are finite. This is not an unrealistic assumption and nor is it limiting. In fact this is the view almost all proposals take in regards to security levels. In the function below, we consider the modes we have already introduced in section 3, namely the *firm* (strict belief), *optimistic* (greedy belief) and *cautious* (conservative belief) modes. We will address the issue of adding user defined belief modes in a later section.

Definition 3.1 Let $\mathbf{R}$ be a set of all possible MLS relations, R be the scheme of any relation r, S be a set of security labels, $\preceq$ and $\prec$ be respectively a partial order and an ordering relation on S, and $\mu = \{\mathit{firm}, \mathit{optimistic}, \mathit{cautious}\}$ be a set of belief modes. Then the belief function β is defined as $\beta : \mathbf{R} \times S \times \mu \rightarrow \mathbf{R}$ such that,

$$\beta(r,s,m) = \left\{ t \;\middle|\; \begin{array}{l} \text{One of the following conditions hold:} \\ \text{- } m = \mathit{firm} \text{ and } t \in r \text{ and } t[TC] = s \\ \text{- } m = \mathit{cautious} \text{ and the following condition holds:} \\ \quad \text{- } \exists u(u \in r, t[TC] = s, u[TC] \preceq s, t[AK, C_{AK}] = u[AK, C_{AK}], \text{ and } \\ \quad \forall i(A_i \in R \text{ and } A_i \notin AK, \exists v(v \in r,\ t[A_i, C_i] = v[A_i, C_i],\ v[TC] \preceq s, \\ \quad v[AK] = t[AK] \text{ and } \neg\exists w(w \in r, w[AK] = t[AK],\ w[TC] \preceq s,\ v[C_i] \prec w[C_i])))). \\ \text{- } m = \mathit{optimistic} \text{ and } t \in r \text{ and } t[TC] \preceq s \end{array} \right.$$

[6]In a syntax that we would like to propose.

[7]Notice that the semantics of belief is not the issue here, rather the process of assigning semantics is. Also note that the use of SQL syntax presented here is just for expository purposes which exactly is not our current mission.

Notice that the above function β will produce the views in figure 6 through 8 except the tuples t_4 and t_5 in figure 7 and t_5 in figure 8 respectively. We will take up the issue of these missing tuples in section 7 again and explain the reason for this behavior. But for now, we just remark that by disallowing these tuples, we are avoiding the generation of the *surprise stories* identified in this paper that compromises the security in MLS databases. Basically, β does not implement the filter function σ in [12] which actually is the source of surprise stories.

4 Contributions of MultiLog and Related Research

The paucity of attempts aimed at developing a logical characterization for MLS models evidences that MLS deductive databases are really at their embryonic state. While there were several proposals such as [17, 6, 2, 10, 11, 25] that addressed the general issue of authorization in a deductive framework, only Cuppens addressed the issue of querying MLS deductive databases [7]. The merits and exigencies of a deductive metaphor of MLS model is eloquently discussed in [26]. In [22] Pernul et al. discuss a prototype developed in $\mathcal{LDL}$ [9] showing that the design process of an MLS relational database, and the assignment of security labels to data and clearances to users may be significantly enhanced using their prototype that is capable of reasoning about the security assignment of the data elements. Their deductive filter prototype is based on a conceptual model developed in [23].

In his proposal [7], Cuppens brings out the inherent difficulty of developing a logic based query language for MLS databases. Although he did not propose linguistic tools or a proof procedure for the lack of a sound axiomatization, to our knowledge, this was the first and only attempt at developing a truly deductive query language for MLS databases until now. While Pernul et al.'s [22] prototyping tool is not a query language, it suggests that a natural and seamless integration of their tool and a MLS deductive database would result in an improved system. In such systems, users will not have to apply a transformation function from relational to deductive representation of their application and vice versa to comprehend and visualize its behavior. It also suggests that the uniformity of the system view could be supported from conceptual design to implementation, only if a logical rendition of MLS model was possible.

Inspired by such necessity and a rich body of existing research in relational counterpart, we make a first-ever attempt to develop a query language, called *MultiLog*, for MLS deductive databases in two steps. In this paper, we develop a foundation for belief reasoning by providing a parametric belief function in the context of MultiLog, and suggest a computational framework by translating MultiLog databases into Datalog. A similar approach has been taken by Jajodia et al [11] to capture multiple access control policies in databases in general. The insight developed in the current research serves as the basis for a complete logical synthesis of MultiLog which we develop in [14] as an orthogonal extension of the work contained in this paper in the direction of F-logic [18]. In [14] we present a complete proof procedure, model theory and fix-point characterization of MultiLog and show that all three characterizations are equivalent. This development is significant from a theoretical standpoint, but we do not attempt to include these results and associated discussion in this paper for the sake of brevity. Complete details may be found in [14].

We make a crucial observation that the user views of the MLS databases at different security levels is mimetic of the notion of *inheritance* in object oriented systems in a slightly elaborate fashion. In both our attempts, we utilized this connection and exploited our experience in dealing with inheritance in logic based systems [5, 15, 13]. The belief function presented in section 3.2 incorporates the results from our work in [15, 13] and extends the idea here further to cater to parametric inheritance.

Our contributions in this paper may be summarized as follows: (i) we propose a F-logic like *query language* (unlike most others) for MLS deductive databases which can be directly used to model applications, (ii) we propose a model for parametric user belief in MLS databases to facilitate ad hoc belief querying and belief speculation, (iii) we provide linguistic instruments for belief querying in multiple modes, (iv) we support user defined belief modes making it possible to tailor the user view as needed, and (v) we show that Datalog is a special case of MultiLog.

5 Overview of MultiLog Language

The language $\mathcal{L}$ of MultiLog is a 7-tuple $\langle\mathcal{P},\mathcal{F},\boldsymbol{A},\mathcal{V},\mathcal{S},\preceq,\mu\rangle$ where (i) $\mathcal{P}$ is an infinite set of predicate names, (ii) $\mathcal{F}$ is an infinite set of function symbols including the symbol `null`, denoted $\perp$, (iii) $\boldsymbol{A}$ is a finite set of attribute names, (iv) $\mathcal{V}$ is a denumerable set of variable names, (iv) $\mathcal{S}$ is a finite set of labels intended to denote the security labels in our language, (v) $\preceq$ is a partial order on the symbols in $\mathcal{S}$ that captures the idea of the hierarchy of security levels, and finally (vii) μ is a finite set of symbols for belief modes. The symbols in $\mathcal{L}$ are pairwise disjoint.

The terms $\mathcal{T}$ of $\mathcal{L}$ are constructed as usual from $\mathcal{F}\cup\mathcal{V}$. Let the ground subset of $\mathcal{T}$ be denoted by $\mathcal{T}^*$ which serves as the constants in our language.

5.1 Formulas and Databases

There are five types of atoms in our language: m-, b-, p-, l- and h-atoms.

- MLS atoms or m-atoms: Let p be a predicate symbol in $\mathcal{P}$ of arity n - denoted p/n, a is an attribute name in $\boldsymbol{A}$, $\boldsymbol{v}$ is a term in $\mathcal{T}$, and $\boldsymbol{s}$ and $\boldsymbol{c}$ are symbols in $\mathcal{S}\cup\mathcal{V}$. Then $\boldsymbol{s}[p(\boldsymbol{k} : a \overset{\boldsymbol{c}}{\rightarrow} \boldsymbol{v})]$ is an m-atom[8]. Intuitively, an m-atom represents a column of a tuple as in MLS relational database counterpart where a is an attribute name, $\boldsymbol{v}$ is a value and $\boldsymbol{s}$ and $\boldsymbol{c}$ are security labels. The label $\boldsymbol{s}$ denotes the security level of the predicate p and mimics the tuple classification TC in MLS relational model.

[8]We also allow a syntactic variant of m-atoms called an m-molecule or m-predicate. An m-predicate has the form $\boldsymbol{s}[p(\boldsymbol{k} : a_1 \overset{\boldsymbol{c}_1}{\rightarrow} \boldsymbol{v}_1,\ldots,a_n \overset{\boldsymbol{c}_n}{\rightarrow} \boldsymbol{v}_n)]$ which is equivalent to the atomic conjunction $\boldsymbol{s}[p(\boldsymbol{k} : a_1 \overset{\boldsymbol{c}_1}{\rightarrow} \boldsymbol{v}_1)]\wedge\ldots\wedge\boldsymbol{s}[p(\boldsymbol{k} : a_n \overset{\boldsymbol{c}_n}{\rightarrow} \boldsymbol{v}_n)]$. Again, an m-predicate may be viewed as a syntactic sugar for classical MLS tuples. The corresponding classical predicate representation of an m-predicate can be written as $p(\boldsymbol{k},a_1,\boldsymbol{v}_1,\boldsymbol{c}_1,\ldots,a_n,\boldsymbol{v}_n,\boldsymbol{c}_n,\boldsymbol{s})$. The only difference with MLS tuples is of course that we include attribute names in our atoms/molecules. This approach was also taken in [15, 18], etc. The advantage of this syntax is that it gives a functional view of predicates and makes the columns position independent. In our discussion and examples that follow, we will freely use either the atomic or the molecular form as the situation demands.

- Believed atoms or b-atoms: Let $\boldsymbol{s}[p(\boldsymbol{k} : a \overset{\boldsymbol{c}}{\rightarrow} \boldsymbol{v})]$ be an m-atom and $m\in\mu$ be a mode of belief by a rational agent. Then $\boldsymbol{s}[p(\boldsymbol{k} : a \overset{\boldsymbol{c}}{\rightarrow} \boldsymbol{v})] \ll m$ is a b-atom. Intuitively, a b-atom says that a rational agent believes $p(\boldsymbol{k} : a \overset{\boldsymbol{c}}{\rightarrow} \boldsymbol{v})$ at level $\boldsymbol{s}$ in a mode m.
- Predicates or p-atoms: If p is a predicate symbol in $\mathcal{P}$ of arity k - denoted p/k, and $\boldsymbol{a}_1,\ldots,\boldsymbol{a}_k$ are terms in $\mathcal{T}$, then $p(\boldsymbol{a}_1,\ldots,\boldsymbol{a}_k)$ is a p-atom. The sense of a p-atom is exactly as the classical logic.
- Level or l-atoms: Let $level/1$ be a distinguished predicate symbol in $\mathcal{P}$, and $\boldsymbol{s}$ be a symbol in $\mathcal{S}\cup\mathcal{V}$. Then $level(\boldsymbol{s})$ is an l-atom. An l-atom declares the existence of a security level in a database $\mathcal{D}$.
- Hierarchy or h-atoms: Let $order/2$ be another distinguished predicate symbol in $\mathcal{P}$, and $\boldsymbol{l}$ and $\boldsymbol{h}$ be two symbols in $\mathcal{S}\cup\mathcal{V}$. Then $order(\boldsymbol{l},\boldsymbol{h})$ is an h-atom. Intuitively an h-atom asserts that the security level $\boldsymbol{l}$ is lower than $\boldsymbol{h}$ and that there are no other $\boldsymbol{i}$ such that $order(\boldsymbol{l},\boldsymbol{i})$ and $order(\boldsymbol{i},\boldsymbol{h})$ hold.

Formulas of $\mathcal{L}$ are defined as usual. A literal is either an atom ($\mathcal{A}$) or the negation of an atom ($\neg\mathcal{A}$). Following the custom in logic programming, we only consider the definite (Horn) clause fragment of our language. A clause in $\mathcal{L}$ is an expression of the form $\mathcal{A}\leftarrow\mathcal{B}_1,\ldots,\mathcal{B}_m$ such that $\mathcal{A}$ and $\mathcal{B}_i$s are atoms of $\mathcal{L}$. If the consequent of a clause is an m-atom, we call the clause an m-clause. Similarly, we define p-, l- and h-clauses. We, however, do not have b-clauses as we do not allow b-atoms to appear in the consequent.

Definition 5.1 (Databases and Queries) A *database* Δ, or equivalently a program $\mathbf{P}$, in MultiLog is an expression of the form $\langle\Lambda,\Sigma,\Pi,\mathcal{Q}\rangle$, where (i) Λ is a set of l- and h-clauses (possibly empty) defining the security levels and inducing a partial order on the levels, (ii) Σ is a set of m-clauses that define the secured data component of Δ, (iii) Π is a set of p-clauses (possibly empty), and finally (iv) $\mathcal{Q}$ is a set of clauses of the form $\leftarrow\mathcal{B}_1,\ldots,\mathcal{B}_m$, called the queries.

While the above definition of programs is acceptable, we consider a restricted subset of MultiLog programs for reasons described below.

Definition 5.2 (Dependency Graph) Let Cl be a clause of the form $\mathcal{A}\leftarrow\mathcal{B}_1,\ldots,\mathcal{B}_m$. Let $\leftharpoonup$ denote the binary relationship *depends on*. For Cl, we say that $\mathcal{A}$ depends on $\mathcal{B}_1,\ldots,\mathcal{B}_m$, denoted $\mathcal{A}\leftharpoonup\mathcal{B}_1,\ldots,\mathcal{A}\leftharpoonup\mathcal{B}_m$. The transitive closure of the relation $\leftharpoonup$ with respect to $\mathcal{A}$ is called the *dependency graph* of $\mathcal{A}$.

We require that similar to MLS relational and classical relational models, MultiLog database m-predicates satisfy several integrity constraints. First, we require that for every m-predicate there is a key attribute AK for which the value is $\boldsymbol{k}$. Hence, there must be an m-atom of the form $\boldsymbol{s}[p(\boldsymbol{k} : a \overset{\boldsymbol{c}}{\rightarrow} \boldsymbol{k})]$. That is for every m-atom of the form $\boldsymbol{s}[p(\boldsymbol{k} : b \overset{\boldsymbol{d}}{\rightarrow} \boldsymbol{v})]$ in a program P[9], we also have $\boldsymbol{s}[p(\boldsymbol{k} : a \overset{\boldsymbol{c}}{\rightarrow} \boldsymbol{k})]$. For such atoms, $\boldsymbol{k}$ is identified as AK, $\boldsymbol{c}$ as $\boldsymbol{c}_{AK}$, $\boldsymbol{s}$ as TC, and for all other atoms for which $\boldsymbol{k}$ is the key, a is identified as A_i, $\boldsymbol{c}$ as C_i and $\boldsymbol{v}$ as A_i in a fashion similar to Jajodia and Sandhu [12].

[9]In fact, in $[\![P]\!]$.

Definition 5.3 (Admissible Databases) Let $[\![\;]\!]$ be the meaning function of a logic program in the classical sense[10]. Let $\Delta = \langle \Lambda, \Sigma, \Pi, \mathcal{Q} \rangle$ be a MultiLog database. We say Δ is *admissible* if, and only if, the following conditions hold:

- for every clause $Cl \equiv \mathcal{A} \leftarrow \mathcal{G} \in \Lambda$, the dependency graph of $\mathcal{A}$ does not contain atoms other than h- or l-atoms[11].
- for every clause $Cl \equiv \mathcal{A} \leftarrow \mathcal{G} \in \Sigma$ and every security label $\boldsymbol{s}$ appearing in $\mathcal{A}$ and $\mathcal{G}$, $\boldsymbol{s}$ is asserted by the meaning of Λ, that is $level(\boldsymbol{s}) \in [\![\Lambda]\!]$.
- The meaning of Λ, i.e., $[\![\Lambda]\!]$, defines a partial order on the set of security levels asserted by Λ.

We also require that every MultiLog database satisfy the core integrity properties defined in Jajodia and Sandhu [12]. Hence, we incorporate the following consistency conditions from [12] as a natural carry over. For an intuitive explanation of these conditions, we refer the readers to [12] and [14].

Definition 5.4 (Consistent Databases) An admissible database $\Delta = \langle \Lambda, \Sigma, \Pi, \mathcal{Q} \rangle$ is called *consistent* if, and only if, the following conditions hold:

- Entity Integrity: Let AK be the apparent key of an m-predicate[12]. The database Δ satisfies entity integrity if, and only if, for every m-predicate in $[\![\Sigma]\!]$ of the form $\boldsymbol{s}[p(\boldsymbol{k} : a_1 \overset{\boldsymbol{c}_1}{\rightarrow} \boldsymbol{v}_1, \ldots, a_n \overset{\boldsymbol{c}_n}{\rightarrow} \boldsymbol{v}_n)]$, the following conditions are true.
  - $\boldsymbol{v}_i \in AK \Rightarrow \boldsymbol{v}_i \neq \bot$[13].
  - $\boldsymbol{v}_i, \boldsymbol{v}_j \in AK \Rightarrow \boldsymbol{c}_i = \boldsymbol{c}_j$, i.e., AK is uniformly classified, and
  - $\boldsymbol{v}_i \notin AK \Rightarrow \boldsymbol{c}_i \succeq \boldsymbol{c}_{AK}$ (where $\boldsymbol{c}_{AK}$ is defined to be the classification of the apparent key AK).
- Null Integrity: The database Δ satisfies null integrity if, and only if, for every m-predicate in $[\![\Sigma]\!]$ of the form $\boldsymbol{s}[p(\boldsymbol{k} : a_1 \overset{\boldsymbol{c}_1}{\rightarrow} \boldsymbol{v}_1, \ldots, a_n \overset{\boldsymbol{c}_n}{\rightarrow} \boldsymbol{v}_n)]$, the following conditions are satisfied.
  - $\boldsymbol{v}_i = \bot \Rightarrow \boldsymbol{c}_i = \boldsymbol{c}_{AK}$, i.e., nulls are classified at the level of the key.
  - We say that an m-predicate of the form $\boldsymbol{s}[p(\boldsymbol{k} : a_1 \overset{\boldsymbol{c}_1}{\rightarrow} \boldsymbol{v}_1, \ldots, a_n \overset{\boldsymbol{c}_n}{\rightarrow} \boldsymbol{v}_n)]$ subsumes another m-predicate $\boldsymbol{s}'[p(\boldsymbol{k}' : a_1 \overset{\boldsymbol{c}'_1}{\rightarrow} \boldsymbol{v}'_1, \ldots, a_n \overset{\boldsymbol{c}'_n}{\rightarrow} \boldsymbol{v}'_n)]$ if for every a_i, either (i) $< \boldsymbol{v}_i, \boldsymbol{c}_i > = < \boldsymbol{v}'_i, \boldsymbol{c}'_i >$, or (ii) $\boldsymbol{v}_i \neq \bot$ and $\boldsymbol{v}'_i = \bot$. We also require that there does not exists two distinct m-predicates in $[\![\Sigma]\!]$ such that they subsume each other.
- Polyinstantiation Integrity: The database Δ satisfies polyinstantiation integrity if, and only if, for every m-predicate of the form $\boldsymbol{s}[p(\boldsymbol{k} : a_1 \overset{\boldsymbol{c}_1}{\rightarrow} \boldsymbol{v}_1, \ldots, a_n \overset{\boldsymbol{c}_n}{\rightarrow} \boldsymbol{v}_n)]$ in $[\![\Sigma]\!]$ we have for all $\boldsymbol{v}_i : \boldsymbol{k}, \boldsymbol{c}_{AK}, \boldsymbol{c}_i \rightarrow \boldsymbol{v}_i$. We say that an m-predicate $\boldsymbol{s}[p(\boldsymbol{k} : a_1 \overset{\boldsymbol{c}_1}{\rightarrow} \boldsymbol{v}_1, \ldots, a_n \overset{\boldsymbol{c}_n}{\rightarrow} \boldsymbol{v}_n)]$ satisfies $\boldsymbol{k}, \boldsymbol{c}_{AK}, \boldsymbol{c}_i \rightarrow \boldsymbol{v}_i$ if, and only if, there does not exist another m-predicate $\boldsymbol{s}'[p(\boldsymbol{k}' : a_1 \overset{\boldsymbol{c}'_1}{\rightarrow} \boldsymbol{v}'_1, \ldots, a_n \overset{\boldsymbol{c}'_n}{\rightarrow} \boldsymbol{v}'_n)]$ in $[\![\Sigma]\!]$ such that $< \boldsymbol{k}, \boldsymbol{c}_{AK}, \boldsymbol{c}_i > = < \boldsymbol{k}, \boldsymbol{c}'_{AK}, \boldsymbol{c}'_i >$ and $\boldsymbol{v}_i \neq \boldsymbol{v}'_i$.

[10] Assigns an Herbrand model to a program P.

[11] Intuitively, the ground closure of Λ does not depend on the clauses defined in other components of Δ.

[12] In this paper, we assume AK is a one attribute key for the sake of simplicity. The case for multi-attribute key is discussed in section 7.

[13] If we assume that the key attribute is attribute a_1, then $\boldsymbol{v}_1 \neq \bot$.

Definition 5.5 (Level of Databases) Let Δ be a consistent database, and u a user with a clearance c. The database Δ is in level c, denoted $\langle \Delta, c \rangle$, if the user u with clearance c accesses Δ.

For the remainder of this paper we assume only consistent databases unless specified otherwise.

Example 5.1 (Encoding Mission in MultiLog) Consider tuple t_1 in figure 1. In MultiLog molecular form, we represent t_1 as a rule (fact) r_1: s[mission(avenger:starship $\overset{s}{\rightarrow}$ avenger; objective $\overset{s}{\rightarrow}$ shipping; destination $\overset{s}{\rightarrow}$ pluto)].

5.2 Operational Semantics

In this section, we discuss the operational semantics of MultiLog by presenting a goal directed sequent style proof system. This style of proof systems has also been adopted in languages such as Miller's module language [20], Contextual Logic Programming [21], SelfLog [3], ORLog [15], etc.

The proof system is defined as a set of properties for the two proof predicates $\vdash$ and $\vdash^{\mu}$. This structure essentially gives rise to a two tier proof system. The proof relation $\vdash$ defines the provability of non b-atoms in general and $\vdash^{\mu}$ defines the provability of b-atoms in one of the modes in μ, i.e., {*cau, opt, fir*}[14]. In fact, $\vdash^{\mu}$ encodes the belief function β discussed in section 3.2, where μ can be any of the defined modes of belief. However, in some cases, the provability relations $\vdash^{\mu}$ and $\vdash$ coincide and are defined in terms of $\vdash$.

Also in this proof system, goals are proved in the context of a user clearance u, called the database level as defined in definition 5.5. The context u may be determined at login time or by using a separate authentication procedure, and the interpreter may use the clearance level u dictated by the user's login id. Using this context, the proof system, in particular, makes sure that provability of m-atoms guard against violation of Bell-La Padula restrictions, i.e., *the simple security property* (no read up) and *the $\star$-property* (no write down) [1, 12].

5.3 Proof Rules

The proof rules given below has the following general form where $\langle \Delta, u \rangle$ is the database at level u (the user level), and *conclusion* is any goal. The goal is provable at level u if the *assumptions* hold at level u. However, the application of the rule depends on the satisfiability of the associated *conditions* on the right.

$$(\text{RULE NAME}) \quad \frac{\langle \Delta, u \rangle \vdash Assumptions}{\langle \Delta, u \rangle \vdash Conclusion} \quad \Big(\; Conditions \; \Big)$$

For the sake of simplicity and without loss of any generality, we assume that all molecular atoms are broken down to atomic forms by replacing such molecules with the conjunction of atomic components and then, if necessary, bringing the program to disjunctive normal form by a preprocessor.

5.4 Remarks on the Proof Theory

The intuitive interpretation of the inference rules in figure 9 are given alongside the rules. However, some additional comments

[14] We are assuming here that {*cau, opt, fir*} are the only belief modes in MultiLog. These are shorthand notations for cautious, optimistic and firm belief modes respectively. The case for user defined belief modes is discussed in section 7.

(REFLEXIVITY)	$\dfrac{\langle \Delta, u\rangle \vdash_{\theta} level(\boldsymbol{l})}{\langle \Delta, u\rangle \vdash_{\theta} \boldsymbol{l} \preceq \boldsymbol{l}}$		Reflexive rule for transitive closure of the *order* relation. The exit rule is covered by DEDUCTION-G.
(TRANSITIVITY)	$\dfrac{\begin{array}{c}\langle \Delta, u\rangle \vdash_{\theta} order(\boldsymbol{l}, H') \\ \langle \Delta, u\rangle \vdash_{\sigma} H' \preceq \boldsymbol{h}[\theta]\end{array}}{\langle \Delta, u\rangle \vdash_{\theta\sigma} \boldsymbol{l} \preceq \boldsymbol{h}}$		Recursive rule for transitive closure of the *order* relation to compute $\boldsymbol{l} \preceq \boldsymbol{h}$.
(EMPTY)	$\dfrac{}{\langle \Delta, u\rangle \vdash_{\epsilon} \Box}$		Empty goal is always true.
(AND)	$\dfrac{\langle \Delta, u\rangle \vdash_{\theta} \mathcal{G}_1 \qquad \langle \Delta, u\rangle \vdash_{\sigma} \mathcal{G}_2[\theta]}{\langle \Delta, u\rangle \vdash_{\theta\sigma} \mathcal{G}_1, \mathcal{G}_2}$		Splits the conjunction and proves individually.
(BELIEF)	$\dfrac{\begin{array}{c}\langle \Delta, u\rangle \vdash^{m}_{\theta} \boldsymbol{l}[p(\boldsymbol{k} : a \overset{\boldsymbol{c}}{\to} \boldsymbol{v})] \\ \langle \Delta, u\rangle \vdash_{\phi} \boldsymbol{l} \preceq u[\theta]\end{array}}{\langle \Delta, u\rangle \vdash_{\theta\phi} \boldsymbol{l}[p(\boldsymbol{k} : a \overset{\boldsymbol{c}}{\to} \boldsymbol{v})] \ll m}$		Fires the proof predicate $\vdash^{m}$ and a success indicates $\boldsymbol{l}[p(\boldsymbol{k} : a \overset{\boldsymbol{c}}{\to} \boldsymbol{v})]$ is believed at level $\boldsymbol{l}$ with security classification $\boldsymbol{c}$, and $\boldsymbol{l}$ being dominated by u.
(DEDUCTION-G)	$\dfrac{\langle \Delta, u\rangle \vdash_{\sigma} \mathcal{G}[\theta]}{\langle \Delta, u\rangle \vdash_{\theta\sigma} \mathcal{A}}$	$\left(\begin{array}{l} \mathcal{A}' \leftarrow \mathcal{G} \in \Delta, \\ \theta = mgu(\mathcal{A}, \mathcal{A}') \\ \mathcal{A} \text{ is p-, h- or l-atom} \end{array}\right)$	Mimics classical deduction for non m-atom goals.
(DEDUCTION-G')	$\dfrac{\begin{array}{c}\langle \Delta, u\rangle \vdash_{\sigma} \mathcal{G}[\theta] \\ \langle \Delta, u\rangle \vdash_{\phi} \boldsymbol{l} \preceq u[\theta\sigma]\end{array}}{\langle \Delta, u\rangle \vdash_{\theta\sigma\phi} \mathcal{A}}$	$\left(\begin{array}{l} \mathcal{A}' \leftarrow \mathcal{G} \in \Delta, \\ \theta = mgu(\mathcal{A}, \mathcal{A}') \\ \mathcal{A} = \boldsymbol{l}[p(\boldsymbol{k} : a \overset{\boldsymbol{c}}{\to} \boldsymbol{v})] \end{array}\right)$	An m-atom goal is provable only if the antecedent of the clause defining the goal is provable and the database level dominates the security level $\boldsymbol{l}$ of the goal and hence does not violate the *no read up* rule.
(DEDUCTION-B)	$\dfrac{\langle \Delta, u\rangle \vdash_{\theta} \mathcal{G}}{\langle \Delta, u\rangle \vdash^{\mu}_{\theta} \mathcal{G}}$	$\left(\begin{array}{l} \mu = fir \text{ or} \\ \mathcal{G} \text{ is a non m-atom} \end{array}\right)$	If $\mathcal{G}$ is conjunctive, or $\Box$, or $\mu = fir$, provability switches to classical deduction. The provability in these cases does not depend on the belief mode.
(DESCEND-O)	$\dfrac{\begin{array}{c}\langle \Delta, u\rangle \vdash_{\sigma} R \preceq \boldsymbol{l} \\ \langle \Delta, u\rangle \vdash_{\theta} R[p(\boldsymbol{k} : a \overset{\boldsymbol{c}}{\to} \boldsymbol{v})][\sigma]\end{array}}{\langle \Delta, u\rangle \vdash^{opt}_{\sigma\theta} \boldsymbol{l}[p(\boldsymbol{k} : a \overset{\boldsymbol{c}}{\to} \boldsymbol{v})]}$		$\boldsymbol{l}[p(\boldsymbol{k} : a \overset{\boldsymbol{c}}{\to} \boldsymbol{v})]$ is optimistically provable at $\boldsymbol{l}$ if $p(\boldsymbol{k} : a \overset{\boldsymbol{c}}{\to} \boldsymbol{v})$ is provable at any lower level R, i.e., $\vdash R[p(\boldsymbol{k} : a \overset{\boldsymbol{c}}{\to} \boldsymbol{v})]$.
(DESCEND-C1)	$\dfrac{\begin{array}{c}\langle \Delta, u\rangle \vdash_{\sigma} order(R, \boldsymbol{l}) \\ \langle \Delta, u\rangle \vdash^{cau}_{\theta} R[p(\boldsymbol{k} : a \overset{\boldsymbol{c}}{\to} \boldsymbol{v})][\sigma]\end{array}}{\langle \Delta, u\rangle \vdash^{cau}_{\sigma\theta} \boldsymbol{l}[p(\boldsymbol{k} : a \overset{\boldsymbol{c}}{\to} \boldsymbol{v})]}$	$\left(\begin{array}{l} \neg\exists\psi, \psi = \\ mgu(< \boldsymbol{l}, p, \boldsymbol{k}, a >, \\ \quad < \boldsymbol{l}', p, \boldsymbol{k}', a >), \\ \mathcal{A}' \leftarrow \mathcal{G} \in \Delta, \\ \mathcal{A}' = \boldsymbol{l}'[p(\boldsymbol{k}' : a \overset{\boldsymbol{b}}{\to} \boldsymbol{v}')] \end{array}\right)$	In the absence of any information at $\boldsymbol{l}$, $\boldsymbol{l}$ cautiously believes $\boldsymbol{l}[p(\boldsymbol{k} : a \overset{\boldsymbol{c}}{\to} \boldsymbol{v})]$ only if an immediate lower level cautiously believes it.
(DESCEND-C2)	$\dfrac{\begin{array}{c}\langle \Delta, u\rangle \vdash_{\sigma} R \preceq \boldsymbol{l} \\ \langle \Delta, u\rangle \vdash_{\theta} R[p(\boldsymbol{k} : a \overset{\boldsymbol{c}}{\to} \boldsymbol{v})][\sigma] \\ \langle \Delta, u\rangle \vdash_{\phi} \boldsymbol{b} \preceq \boldsymbol{c}[\sigma\theta] \\ \langle \Delta, u\rangle \vdash_{\psi} \boldsymbol{l}[p(\boldsymbol{k} : a \overset{\boldsymbol{b}}{\to} \boldsymbol{v}')][\sigma\theta\phi]\end{array}}{\langle \Delta, u\rangle \vdash^{cau}_{\sigma\theta\phi\psi} \boldsymbol{l}[p(\boldsymbol{k} : a \overset{\boldsymbol{c}}{\to} \boldsymbol{v})]}$		The information at level $\boldsymbol{l}$ is rejected and a lower level information is accepted at $\boldsymbol{l}$ which has a higher security level $\boldsymbol{c}$, and hence, is more secure. Note that the security label at the lower label cannot be higher than $\boldsymbol{l}$ itself.
(DESCEND-C3)	$\dfrac{\begin{array}{c}\langle \Delta, u\rangle \vdash_{\sigma} \boldsymbol{l}' \preceq \boldsymbol{l} \\ \langle \Delta, u\rangle \vdash_{\theta} \boldsymbol{c} \preceq \boldsymbol{b}[\sigma] \\ \langle \Delta, u\rangle \vdash_{\phi} \boldsymbol{l}[p(\boldsymbol{k} : a \overset{\boldsymbol{c}}{\to} \boldsymbol{v})][\sigma\theta]\end{array}}{\langle \Delta, u\rangle \vdash^{cau}_{\sigma\theta\phi} \boldsymbol{l}[p(\boldsymbol{k} : a \overset{\boldsymbol{c}}{\to} \boldsymbol{v})]}$	$\left(\begin{array}{l} \neg\exists\psi, \psi = \\ mgu(< p, \boldsymbol{k}, a >, \\ \quad < p, \boldsymbol{k}', a >), \\ \mathcal{A}' \leftarrow \mathcal{G} \in \Delta, \\ \mathcal{A}' = \boldsymbol{l}'[p(\boldsymbol{k}' : a \overset{\boldsymbol{b}}{\to} \boldsymbol{v}')] \end{array}\right)$	The information at level $\boldsymbol{l}$ is the most secured compared to any lower level information, if it exists, i.e., has the highest security level $\boldsymbol{c}$.
(DESCEND-C4)	$\dfrac{\langle \Delta, u\rangle \vdash_{\theta} \boldsymbol{l}[p(\boldsymbol{k} : a \overset{\boldsymbol{c}}{\to} \boldsymbol{v})]}{\langle \Delta, u\rangle \vdash^{cau}_{\theta} \boldsymbol{l}[p(\boldsymbol{k} : a \overset{\boldsymbol{c}}{\to} \boldsymbol{v})]}$	$\left(\neg\exists R, order(R, \boldsymbol{l})[\theta] \in \Delta \right)$	$\boldsymbol{l}$ is the lowest level and hence $\boldsymbol{l}[p(\boldsymbol{k} : a \overset{\boldsymbol{c}}{\to} \boldsymbol{v})]$ is the most secure information.

Figure 9: MultiLog proof system.

$$\Lambda_{D_1} := \left|\begin{array}{ll} r_1: & level(u). \\ r_2: & level(c). \\ r_3: & level(s). \\ r_4: & order(u,c). \\ r_5: & order(c,s). \end{array}\right. \qquad \Sigma_{D_1} := \left|\begin{array}{ll} r_6: & u[p(k:a \overset{u}{\rightarrow} v)]. \\ r_7: & c[p(k:a \overset{c}{\rightarrow} t)] \leftarrow q(j). \\ r_8: & s[p(k:a \overset{u}{\rightarrow} v)] \leftarrow c[p(k:a \overset{c}{\rightarrow} t)] \ll cau. \end{array}\right. \qquad \Pi_{D_1} := \left|\begin{array}{ll} r_9: & q(j). \end{array}\right.$$

$$\mathcal{Q}_{D_1} := \left|\begin{array}{ll} r_{10}: & ?\ c[p(k:a \overset{u}{\rightarrow} v)] \ll opt. \end{array}\right.$$

Figure 10: Database D_1.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\langle D_1,c\rangle \vdash_\epsilon \Box}\text{(empty)}}{\langle D_1,c\rangle \vdash_{R/u} R \preceq c}\text{(deduction-g)} \qquad \dfrac{\dfrac{}{\langle D_1,c\rangle \vdash_\epsilon \Box}\text{(empty)}}{\langle D_1,c\rangle \vdash R[p(k:a \overset{u}{\rightarrow} v)][R/u]}\text{(deduction-g)}}{\langle D_1,c\rangle \vdash^{opt}_{\{R/u\}} c[p(k:a \overset{u}{\rightarrow} v)]}\text{(descend-o)} \qquad \dfrac{\dfrac{}{\langle D_1,c\rangle \vdash_\epsilon \Box}\text{(empty)}}{\langle D_1,c\rangle \vdash_\epsilon c \preceq c}\text{(deduction-g)}}{\langle D_1,c\rangle \vdash_{\{R/u\}} c[p(k:a \overset{u}{\rightarrow} v)] \ll opt}\text{(belief)}$$

Figure 11: A proof tree for $\langle D_1, c\rangle \vdash_{\{R/u\}} c[p(k:a \overset{u}{\rightarrow} v)] \ll opt$.

are warranted. The EMPTY, AND and DEDUCTION-G are natural carryovers from classical logic which every interpreter has. The DEDUCTION-G' rule, however, is peculiar to MultiLog. This rule captures the idea that an m-atom is true via deduction if, and only if, the security level, namely $\boldsymbol{l}$, is dominated by the database level. Similarly, the BELIEF rule enforces the no read up policy similar to DEDUCTION-G', but now via the $\vdash^m$ proof rules. The proof rules DESCEND-O through DESCEND-C4 implement the belief function β discussed in section 3.2. In particular, rule DESCEND-O implements the *opt* visibility, while DESCEND-C1 through DESCEND-C4 capture *cau* visibility. The *fir* visibility is trivially captured by DEDUCTION-G' rule. The provability in any mode μ ($\vdash^\mu$) is equivalent to the general provability ($\vdash$) if the goal $\mathcal{G}$ is conjunctive, empty, or a b-atom. This observation leads to the inclusion of the DEDUCTION-B rule.

As usual, and as in [21, 20, 4], a proof for $\langle \Delta, u\rangle \vdash \mathcal{G}\theta$ is a tree, called the *proof tree*, rooted at $\langle \Delta, u\rangle \vdash_\theta \mathcal{G}$ with internal nodes that are instances of one of the above rules, and with leaf nodes that are labeled with the figure EMPTY. The *height* of a proof is the maximum of the number of nodes in all the branches in the proof tree, and the *size* of a proof is the number of nodes in the proof tree.

Example 5.2 Consider the database D_1 in figure 10, and query r_{10}. For this query, let us assume that the database is at level c. The procedure succeeds in constructing a successful proof tree (shown in figure 11), hence a proof. Observe that the leaf nodes are instances of the proof rule EMPTY, indicating a successful proof as defined in section 5.4.

6 MultiLog Front-end for CORAL

We now present a MultiLog front-end architecture for CORAL deductive database. While this front-end can be viewed as an implementation scheme for MultiLog, a more direct implementation is also possible. As we noted earlier, a detailed discussion on the model theory and fix-point characterization of MultiLog and their equivalence to the operational semantics presented here may be found in [14]. The front-end has the following architecture.

6.1 Reduction to CORAL

The reduction proceeds in three steps. First, we break the molecular formulas into atomic conjunctions and bring the program to disjunctive normal form. Then the resulting program is encoded into a CORAL program by applying a suitable translation function τ that incorporates the user's clearance level into the translated program too. Then, the query is executed on the encoded program by adding the MultiLog interpreter that augments the CORAL interpreter with the rules implementing the additional proof rules (the proof predicates) of MultiLog. As far as the users are concerned, the results are still given as a set of binding to the query variables, and hence the reduction process and the use of CORAL as a back-end remain transparent to the users.

We are now ready to define an algorithm to reduce every MultiLog program to CORAL. This requires us to develop a translation function τ that will map every MultiLog expression to CORAL expressions. We proceed as follows.

Given any MultiLog expression ϕ, its encoding into CORAL, denoted $\phi^\star$, is given by the following recursive transformation rules. In the following, τ is an identity function on the terms and symbols in MultiLog.

- Encoding of complex formulas:
  - $\tau(\mathcal{A} \leftarrow \mathcal{B}_1, \ldots, \mathcal{B}_m) = \tau(\mathcal{A}) \leftarrow \tau(\lambda(\mathcal{B}_1, u)), \ldots, \tau(\lambda(\mathcal{B}_m, u))$
- Encoding of atomic MultiLog formulas (given case by case):
  - $\tau(\boldsymbol{l}[p(\boldsymbol{k}: a \overset{\boldsymbol{c}}{\rightarrow} \boldsymbol{v})]) = \mathsf{rel}(p, \boldsymbol{k}, a, \boldsymbol{v}, \boldsymbol{c}, \boldsymbol{l})$.
  - $\tau(\boldsymbol{l}[p(\boldsymbol{k}: a \overset{\boldsymbol{c}}{\rightarrow} \boldsymbol{v})] \ll m) = \mathsf{bel}(p, \boldsymbol{k}, a, \boldsymbol{v}, \boldsymbol{c}, \boldsymbol{l}, m)$.
  - $\tau(p(\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n)) = p(\boldsymbol{a}_1, \ldots, \boldsymbol{a}_n)$.
  - $\tau(level(\boldsymbol{l})) = level(\boldsymbol{l})$.
  - $\tau(order(\boldsymbol{l}, \boldsymbol{h})) = order(\boldsymbol{l}, \boldsymbol{h})$.

a_1 dominate(X, Y) ← order(X, Y).
a_2 dominate(X, X) ← level(X).
a_3 dominate(X, Y) ← order(X, Z), dominate(Z, Y).

a_4 bel(P, K, A, V, C, H, fir) ← rel(P, K, A, V, C, H).
a_5 bel(P, K, A, V, C, H, opt) ← rel(P, K, A, V, C, L), dominate(L, H).
a_6 bel(P, K, A, V, C, H, cau) ← rel(P, K, A, V, C, H), ¬order(L, H).
a_7 bel(P, K, A, V, C, H, cau) ← order(L, H), ¬rel(P, K, A, V', C', H), bel(P, K, A, V, C, L, cau).
a_8 bel(P, K, A, V, C, H, cau) ← rel(P, K, A, V', C', H), rel(P, K, A, V, C, L), dominate(L, H), dominate(C', C).
a_9 bel(P, K, A, V, C, H, cau) ← rel(P, K, A, V, C, H), ¬rel(P, K, A, V', C', L), dominate(L, H), dominate(C, C').

Figure 12: MultiLog Inference Engine.

- Encoding of reduction expressions (given case by case):
  - $\tau(\lambda(\mathcal{B}, u)) = \tau(\mathcal{B})$ if $\mathcal{B}$ is a p-, l-, or h-atom.
  - $\tau(\lambda(\mathcal{B}, u)) = \tau(\mathcal{B}), \tau(\boldsymbol{l} \preceq u), \tau(\boldsymbol{c} \preceq u)$ if $\mathcal{B} = \boldsymbol{l}[p(\boldsymbol{k} : a \overset{\boldsymbol{c}}{\rightarrow} \boldsymbol{v})]$, or $\mathcal{B} = \boldsymbol{l}[p(\boldsymbol{k} : a \overset{\boldsymbol{c}}{\rightarrow} \boldsymbol{v})] \ll m$.
  - $\tau(\boldsymbol{l} \preceq \boldsymbol{h}) = \mathsf{dominate}(\boldsymbol{l}, \boldsymbol{h})$.

6.2 The MultiLog Engine

Since CORAL has a classical inference engine, we must augment provability in CORAL with MultiLog provability relations so that together they achieve MultiLog functionality. Since the rules EMPTY, AND and DEDUCTION-G are part of CORAL engine, we encode most of the remaining rules and add to every program as a set of axioms $\mathbf{A}$. Hence a reduced database Δ_r is a pair $\langle \tau(\Delta), \mathbf{A}\rangle$. We chose to implement the set of axioms $\mathbf{A}$ directly in CORAL as shown in figure 12 since this set is an invariant for every reduced program. It may be noted here that the axioms contain negation, while the programs do not, and that the axioms are actually stratified. This should not be confused with programs with negation as the axioms only implement the MultiLog inference engine.

Notice that the mapping for the translation from the MultiLog proof rules to the axioms in the inference engine is not one to one. Apart from the comments above, we also do not implement an axiom for BELIEF, DEDUCTION-G', and DEDUCTION-B. The reason for this deviation is that the implementation of BELIEF and DEDUCTION-G' are part of the encoding process through λ where we add two subgoals of the form $\boldsymbol{l} \preceq u$ and $\boldsymbol{c} \preceq u$ for every m- and b-atom in the body of a clause (i.e., queries). The reason for this approach is that we do not model users and their clearances as first class entities in the database and hence, the level of the database we are interested in must be determined at the compile time since the reduced CORAL program cannot enforce the user specific view of the database. The rule DEDUCTION-B is a by product of the encoding style and the CORAL functionality.

It is easy to establish the correctness of the reduction through the following theorem.

Theorem 6.1 (Correctness of Reduction) Let $\langle \Delta, u\rangle$ be the database Δ at level u, and $\tau(\Delta)$ be its encoding in CORAL. Let $\mathcal{M}$ be the model for the reduced database $\Delta_r = \langle \tau(\Delta), \mathbf{A}\rangle$. Then

$$\langle \Delta, u\rangle \vdash_\theta G \Longleftrightarrow \mathcal{M} \models \tau(G)[\theta]$$

for any MultiLog goal G.

Proof Sketch: By showing that if the proof tree in MultiLog has height k, then the goal $\tau(G)[\theta]$ is computed at step k by the fix-point operator $\mathbf{T}_{\Delta_r}$ for Δ_r and showing that the model $\mathcal{M} = lfp(\mathbf{T}_{\Delta_r})$.

It is somewhat easy to make the observation that for any MultiLog program $\Delta = \langle \Lambda, \Sigma, \Pi, \mathcal{Q}\rangle$, if the Λ and Σ components are empty and the $\mathcal{Q}$ component do not contain any m- or b-atoms, Δ degenerates into a Datalog program. In this case, the proof trees generated by the interpreter for any successful proof will contain instances of the proof rules EMPTY, AND, and DEDUCTION-G which are exactly like classical proof trees for Datalog. The following proposition follows naturally.

Proposition 6.1 (Extension) Let $\vdash^\star$ be the proof predicate for Datalog, and $\mathbf{P}$ be a Datalog program. Then for any Datalog goal $\mathcal{G}$ we have

$$\mathbf{P} \vdash^\star \mathcal{G}[\theta] \Longleftrightarrow \langle \Delta, u\rangle \vdash \mathcal{G}[\phi]$$

where $\Delta = \langle \emptyset, \emptyset, \mathbf{P}, \{\leftarrow \mathcal{G}\}\rangle$ and u is any user level (perhaps system).

Proof Sketch: By showing that the proof rules for $\vdash^\star$ is a subset of $\vdash$ and that the proof trees are identical and yield identical bindings for $\mathcal{G}$, i.e., $\theta = \phi$.

7 MultiLog System Implementation Issues

It is perhaps desirable to avoid any mention of the security level of the data elements or the tuples, or the clearance level of users altogether and present an illusion of a classical relation to users. This is probably the motivation for the works reported in [7, 16, 27] where the authors avoid any mention of the attribute or the tuple classification altogether. This can be achieved in MultiLog by inserting *don't care* variables "_" in place of missing level information in formulas.

We have not incorporated the *filter* function σ discussed in [12] in relation to *inter instance integrity* for several reasons. Firstly, it is still unclear if it makes any sense in a logic based framework such as ours. As we pointed out in section 3, the inheritance of null values gives rise to the unwanted property of surprise stories. We believe that Jajodia and Sandhu incorporated this aspect in their model for technical reasons only as it was unavoidable in their framework. Secondly, we have only considered inheritance of tuples from lower level

(FILTER)	$\dfrac{\begin{array}{c}\langle \Delta, u\rangle \vdash_{\phi} \boldsymbol{l} \preceq R \quad \langle \Delta, u\rangle \vdash_{\sigma} \boldsymbol{c} \preceq \boldsymbol{l}[\phi] \\ \langle \Delta, u\rangle \vdash_{\theta} R[p(\boldsymbol{k} : a \overset{\boldsymbol{c}}{\rightarrow} \boldsymbol{v})][\phi\sigma]\end{array}}{\langle \Delta, u\rangle \vdash_{\phi\sigma\theta} \boldsymbol{l}[p(\boldsymbol{k} : a \overset{\boldsymbol{c}}{\rightarrow} \boldsymbol{v})]}$	A lower level inherits part of the higher level tuples with data elements whose security level is dominated by the current lower level.
(FILTER-NULL)	$\dfrac{\begin{array}{c}\langle \Delta, u\rangle \vdash_{\phi} \boldsymbol{l} \preceq R \quad \langle \Delta, u\rangle \vdash_{\sigma} \boldsymbol{l} \preceq \boldsymbol{c}[\phi] \\ \langle \Delta, u\rangle \vdash_{\theta} R[p(\boldsymbol{k} : a \overset{\boldsymbol{c}}{\rightarrow} \boldsymbol{v})][\phi\sigma]\end{array}}{\langle \Delta, u\rangle \vdash_{\phi\sigma\theta} \boldsymbol{l}[p(\boldsymbol{k} : a \overset{\boldsymbol{l}}{\rightarrow} \perp)]}$	Inherit *null* only if the data element's security level dominates the current lower level.
(USER-BELIEF)	$\dfrac{\begin{array}{c}\langle \Delta, u\rangle \vdash_{\phi} \mathsf{bel}(p, \boldsymbol{k}, a, \boldsymbol{v}, \boldsymbol{c}, \boldsymbol{l}, m) \\ \langle \Delta, u\rangle \vdash_{\sigma} \boldsymbol{l} \preceq u[\phi]\end{array}}{\langle \Delta, u\rangle \vdash_{\phi\sigma} \boldsymbol{l}[p(\boldsymbol{k} : a \overset{\boldsymbol{c}}{\rightarrow} \boldsymbol{v})] \ll m}$	Just copy the proof for bel rule if the depth of b-atom is dominated by u.

Figure 13: Additional rules for MultiLog proof system to incorporate filtering, filtering with *null* and user defined belief function.

to upper levels in different modes of belief. If we were to accommodate this feature, a similar approach may be adopted to inherit formulas in the reverse direction. This extension can be handled orthogonally. To give an idea, the rule FILTER in figure 13 can be used to achieve this functionality[15].

A final reason for not allowing the filter function is as follows. Consider the relations in figures 7 and 8 corresponding to the optimistic and cautious view of the *Mission* relation at level C. Notice in particular that the tuples t_4 and t_5 contain null values. These tuples are the result of the application of the above mentioned filter function as these tuples have migrated from a higher security level to C. Now, if we simultaneously allow filtering and molecular programming as a syntactic variant of atomic programming, we are faced with an implementation problem if we are to keep the current proof system. Consider the proof for

$\langle \Delta, c\rangle \vdash c[mission(phantom : starship \overset{\boldsymbol{c}}{\rightarrow} phantom;$
$objective \overset{\boldsymbol{d}}{\rightarrow} X; destination \overset{\boldsymbol{e}}{\rightarrow} Y)],$

or

$\langle \Delta, c\rangle \vdash c[mission(phantom : starship \overset{\boldsymbol{c}}{\rightarrow} phantom;$
$objective \overset{\boldsymbol{d}}{\rightarrow} X; destination \overset{\boldsymbol{e}}{\rightarrow} Y)] \ll cau,$

or the version

$\langle \Delta, c\rangle \vdash c[mission(phantom : starship \rightarrow phantom;$
$objective \rightarrow X; destination \rightarrow Y)] \ll opt$

using *don't care* variables as discussed above. All these queries fail as the atomic conjunctions fail due to non-availability of objective and/or destination information. Since we do not have the filter function[16], our current system does not fail for such reasons as these tuples are not supported in our model and are never a possibility. But if we were to support both inter instance integrity and molecular programming, we can proceed by adding one more proof rule FILTER-NULL for the filter function as shown in figure 13.

For the sake of simplicity of presentation, we have also assumed single attribute keys throughout this paper. This restriction can also be relaxed in an actual implementation without much difficulty. An F-logic [18] like approach may be adopted to allow set values of the form $\boldsymbol{l}[p(\boldsymbol{k} : a \overset{\boldsymbol{c}}{\twoheadrightarrow} \boldsymbol{v})]$ for key attributes while enforcing functionality requirement on the others, and by adjusting the proof rules accordingly.

A final note about the possibility of user defined belief function in MultiLog. Such user tailored function is always possible. This can be achieved by simply defining rules using a distinguished predicate, say bel, with a predetermined list of arguments and associated meanings. Then, we could proceed to add a proof rule USER-BELIEF to copy this predicate as a proof for a b-atom as shown in figure 13.

It should be pointed out here that this approach to user defined belief function is robust. That is, it does not pose any security threat to the system and does not break down the protocol. This is simply because the provability, and thus the satisfaction in interpretation structures, of m-atoms stays unchanged.

8 Conclusion

To our knowledge, MultiLog is the first logic based query language for MLS databases. It provides support for multiple belief models and ad hoc belief reasoning. It is free from security breach such as surprise stories identified in this paper. It also supports the possibility of tailoring the belief functions according to the application needs making it incremental.

We have shown that MultiLog is a natural extension of Datalog. The scheme presented here for the implementation of MultiLog based on rewriting into CORAL has been shown to be consistent. Several implementation issues have also been discussed. While it is possible to write programs in Datalog that simulate the MultiLog behavior, such programs, nonetheless, are Datalog programs and does not provide the level of abstraction MultiLog does. In such programs, users must apply the transformation function τ in their mental model of the database and be very judicious. Moreover, insuch programs the multilevel abstraction is lost or hidden making it difficult to reason with the program and debug.

[15]In fact, we will have to do much more. In addition to this rule we will have to make sure that the rest of the tuples get inherited only if the keys of the corresponding tuples do so too. We will also have to worry about the effect of this inheritance on the belief modes and the notion of *subsumption* discussed in [12].

[16]This means our databases do not satisfy the inter instance integrity discussed in [12] and we do not think it is detrimental to our system. But if needed we can incorporate this feature without any trouble as discussed in this section.

While this paper deals with the implementation aspects of MultiLog, and the model for a parametric belief function, the theoretical foundations of MultiLog have been developed in [14] where we present a sound and complete proof procedure with respect to the model theory and fix point semantics of MultiLog. We utilized a crucial connection between the concept of inheritance in object-oriented systems and the views at different levels in a MLS database. This connection helped us to develop the logical semantics presented here and in [14]. We believe Cuppens' difficulty in developing a complete axiomatization could have been removed if this connection was established.

As future research, we would like to investigate further the issues raised in section 7. We also plan to run a comparison with existing relational MLS implementations and MultiLog. These are some of the issues we seek to investigate in the immediate future.

Acknowledgment: Research supported in part by a grant from the Computer Science Department, Mississippi State University, USA. The author would like to thank Durriya Meer of Wright State University, USA for giving helpful comments to improve the presentation of the initial version of the paper.

References

[1] D. E. Bell and L. J. La Padula. Secure computer systems: Unified exposition and multics interpretation. Technical Report ESD-TR-75-306, The MITRE Corporation, Bedford, MA, March 1976.

[2] P. Bonatti, S. Kraus, and V. S. Subrahmanian. Foundations of secure deductive databases. *IEEE Transactions on Knowledge and Data Engineering*, 7(3):406–422, 1995.

[3] M. Bugliesi. A declarative view of inheritance in logic programming. In K. Apt, editor, *Proc. Joint Int. Conference and Symposium on Logic Programming*, pages 113–130. The MIT Press, 1992.

[4] M. Bugliesi and H. M. Jamil. A logic for encapsulation in object oriented languages. In M. Hermenegildo and J. Penjam, editors, *Proceedings of the 6th International Symposium on Programming Language Implementation and Logic Programming (PLILP)*, pages 213–229, Madrid, Spain, 1994. Springer-Verlag. LNCS 844.

[5] M. Bugliesi and H. M. Jamil. A stable model semantics for behavioral inheritance in deductive object oriented languages. In G. Gottlob and M. Y. Vardi, editors, *Proceedings of the 5th International Conference on Database Theory (ICDT)*, pages 222–237, Prague, Czech Republic, 1995. Springer-Verlag. LNCS 893.

[6] K. S. Candan, S. Jajodia, and V. S. Subrahmanian. Secure mediated databases. In *ICDE Proc.*, pages 35–55, 1996.

[7] F. Cuppens. Querying a multilevel database: A logical analysis. In *Proc of the VLDB Conference*, Mumbai, India, August 1996.

[8] D. E. Denning, T. F. Lunt, R. R. Schell, M. Heckman, and W. R. Shockley. A multilevel relational data model. In *Proc. of the IEEE Symposium on Security and Privacy*, pages 220–234. IEEE Computer Society Press, 1987.

[9] D. Chimenti et. al. The $\mathcal{LDL}$ system prototype. *IEEE Journal on Data and Knowledge Engineering*, 2(1):76–90, 1990.

[10] S. Jajodia, P. Samarati, and V. S. Subrahmanian. A logical language for expressing authorizations. In *Proc. IEEE Symp. on Security and Privacy*, pages 31–42, Oakland, CA, May 1997.

[11] S. Jajodia, P. Samarati, V. S. Subrahmanian, and E. Bertino. A unified framework for enforcing multiple access control policies. In *SIGMOD Proc.*, pages 474–485, 1997.

[12] S. Jajodia and R. Sandhu. Toward a multilevel secure relational data model. In *Proc. of the Conf. on Management of Data*, pages 50–59, Denver, CO, May 1991. ACM Press.

[13] H. M. Jamil. Implementing abstract objects with inheritance in Datalog$^{\mathrm{neg}}$. In *Proceedings of the 23rd International Conference on Very Large Databases (VLDB)*, pages 56–65, Athens, Greece, 1997.

[14] H. M. Jamil. A logical foundation for mls deductive databases. Technical report, Department of Computer Science, Mississippi State University, USA, November 1998. Submitted for publication.

[15] H. M. Jamil and L. V. S. Lakshmanan. A declarative semantics for behavioral inheritance and conflict resolution. In John Lloyd, editor, *Proceedings of the 12th International Logic Programming Symposium*, pages 130–144, Portland, Oregon, December 1995. MIT Press.

[16] N. A. Jukic and S. V. Vrbsky. Asserting beliefs in mls relational models. In *Sigmod Record*, pages 30–35, Ithaca, NY, 1997. ACM Press.

[17] V. Kessler and G. Wedel. Autlog – an advanced logic of authentication. Manuscript.

[18] M. Kifer, G. Lausen, and J. Wu. Logical Foundations for Object-Oriented and Frame-Based Languages. *Journal of the Association of Computing Machinery*, 42(3):741–843, July 1995.

[19] T. F. Lunt, D. E. Denning, R. R. Schell, M. Heckman, and W. R. Shockley. The seaview security model. *IEEE Transactions on Software Engineering*, 16(6):593–607, 1990.

[20] D. Miller. A Logical Analysis of Modules in Logic Programming. *Journal of Logic Programming*, 6(1/2):79–108, January/March 1989.

[21] L. Monteiro and A. Porto. Contextual Logic Programming. In *6th ALP Intl. Conf. on Logic Programming*, 1989.

[22] G. Pernul, W. Winiwarter, and A. M. Tjoa. The deductive filter approach to mls database prototyping. In *Proc. of the 9th Annual Computer Security Applications Conference*, Orlando, FL, December 1993.

[23] G. Pernul, W. Winiwarter, and A. M. Tjoa. The entity relationship model for multilevel security. In *Proc. of the 12th Intl. Conf. on Entity-Relationship Approach*, Dallas, TX, December 1993.

[24] K. Smith and M. Winslett. Entity modeling in the MLS relational model. In *Proc. of the 18th VLDB Conference*, pages 199–210, Vancouver, BC, August 1992.

[25] A Spalka. Fundamental forms of confidentiality in deductive databases. Manuscript.

[26] W. Winiwarter. Why is deduction required for database systems ? - some case studies. In *Proc. of the 2nd Data Engineering Forum*, Tokyo, Japan, November 1995.

[27] M. Winslett, K. Smith, and X. Qian. Formal query languages for secure relational databases. *ACM Transactions on Database Systems*, 19(4):626–662, December 1994.