A Formal Analysis of the Lightweight Directory Access Protocol

Fang Wei and Georg Lausen

University of Freiburg, Institute for Computer Science, Germany
{fwei,lausen}@informatik.uni-freiburg.de

Abstract. LDAP (Lightweight Directory Access Protocol) directories are being widely used on the Web, for white pages information, user profiles, etc. The advantages LDAP offers are (i) the support for highly distributed data on the Web while still keeping a uniform data model; (ii) the flexibility of a semi-structured data model, i.e. a flexible data type definition enabling the presentation and manipulation of heterogeneous data entries in a natural manner. Although many implementations of the LDAP protocol exist, the still lacking logical formalization prohibits a formal analysis and makes it difficult to make use of the numerous results developed for relational databases. In this paper, we give a first-order logic semantics of LDAP and discuss the expressive power of LDAP. In particular, schema typing constraints are interpreted as semantic integrity constraints. We apply our framework to the containment problem of LDAP queries with schema constraints; we reduce this problem to the containment problem of Datalog in the presence of integrity constraints.

1 Introduction

LDAP (Lightweight Directory Access Protocol) directories are being widely used on the Web. Besides the traditional directory application like white pages information, user profiles, etc, it has been deployed in a number of commercial directory server implementations [12] (also see [11] for a survey). The Internet Engineering Task Force (IETF) has standardized the Lightweight Directory Access Protocol (LDAPv3) for modeling and querying network directory information [22].

An LDAP-based network directory can be viewed as a highly distributed database, in which the directory entries are organized in a hierarchical namespace and can be accessed using database-style search functions. The advantages LDAP offers are (i) the support for highly distributed data on the Web while still keeping a uniform data model; (ii) the flexibility of a semi-structured data model, i.e. a flexible data type definition enabling the presentation and manipulation of heterogeneous data entries in a natural manner. Due to the similarities of the data model of LDAP and XML, there have been works on the transformation from XML to LDAP [15].

In spite of the various implementations of the LDAP protocol, the still lacking logical formalization prohibits a formal analysis and makes it difficult to make use of the numerous results developed for relational databases. In this paper, we seek to analyze the first order semantics of a full-fledged LDAP query language and make the following contribution:

- We give a formal description of the LDAP directory data model in Section 2, analyze the expressive power of the LDAP query language, and present a complete algorithm to transform LDAP queries to *Datalog*.
- Query optimization is of general interest in traditional databases. Since determining equivalence of queries is one of the most fundamental query optimization problems, we demonstrate the query equivalence or query containment problem of LDAP in the first order logic semantics (Section 2.3).
- Integrity constraints are crucial in semantic query optimization and view updating. In Section 3 we transform the LDAP schema information into a form of generalized integrity constraints and tackle the containment problem in the presence of such integrity constraints.

Section 4 compares our results to previous work in the literature, and finally Section 5 concludes this paper.

2 LDAP Directory Model

The formal description of LDAP directory can be found in [4, 7] and the informal one in [10]. However, the form and syntax of these descriptions make them extremely difficult to be directly applicable to the techniques and results found in the literature. Therefore, we need to abstract the peculiarities of the LDAP syntax (Section 2.1), to then reconstruct the model in standard logical terminology (Section 2.2).

2.1 General Model

An LDAP directory consists of the following two main components:

Directory Schema: Defines a finite set of classes, attributes, and types. Each attribute has exactly one type, and each class specifies a set of `required` and `allowed` attributes.

Directory Instance: Contains a finite set of entries organized in a forest, where each entry
  - belongs to at least one class,
  - has a non-empty set of (possibly) multivalued attribute-value pairs that conform to the schema definition,

- defines at least attributes *oc* and *dn*, where *oc* determines what classes the entry belongs to, and *dn* provides a unique distinguished name for the entry,
- is placed in the instance hierarchy based on its distinguished name.

In order to show an example of LDAP schema and instance, we take directly from [7]. All the examples refer to the schema given in Table 1, which shows how an LDAP schema is defined. Figure 1 gives an LDAP instance; note each entry has a unique *distinguished name* (DN) which acts as a path expression in the semi-structured data model. The directory describes the whole organization, contains sub-directories for e.g. persons, groups, etc. The bottom entry describes a person, and it is the instance of the class inetOrgPerson and external which is defined in Table 1 (Here the attribute oc means ObjectClass). Furthermore, it has a set of (attribute, value) pairs. Note that multi-value attributes are allowed here, e. g. the entry has two values for the attribute sn, cluet and verduron.

Table 1. Schema of an example LDAP directory

class	external	inetOrgPerson	internal	residentialPerson
requires	oc, mail, uniqueId	oc,sn,cn	oc,uniqueId,cn	oc,sn,cn,locality
allowes	cn	tel,mail,age	mail,sn,gender,locality,tel	tel

In addition, LDAP offers a rather limited query facility based on filter definitions. A query is defined by a filter, which consists of the following four components:

Base: The *distinguished name* (DN) of the entry in the directory instance where the search will start. The DN is a sequence $[s_1, ..., s_n]$, where each of the s_i is an attribute-value pair (attr, val). The first pair, s_1, is called *relative distinguished name* (RDN), which distinguishes the entry from its sibling entries. For example, the DN of the bottom entry of Figure 1 is: [(name,cluet), (ou,people), (o,att.com)], its RDN is (name, cluet).

Scope: Can be base [1], if the search is to be restricted to just the first node, one, if only the first level of nodes is to be searched, or sub, if all nodes under the base should be considered by the filter expression.

Filter Expression: Defined as the boolean combination of atomic filters of the form (a *op* t), where:
- a is an attribute name;
- *op* is a comparison operator out of the set $\{=, <, \leq, >, \geq\}$;
- t is an attribute value, or $*$, used to test for existence of an attribute.

The boolean operators are `and` (&), `or` (|) and `not` (!).

[1] It is different from the **Base** above. We write from now on **Base** for the root entry and base for the scope.

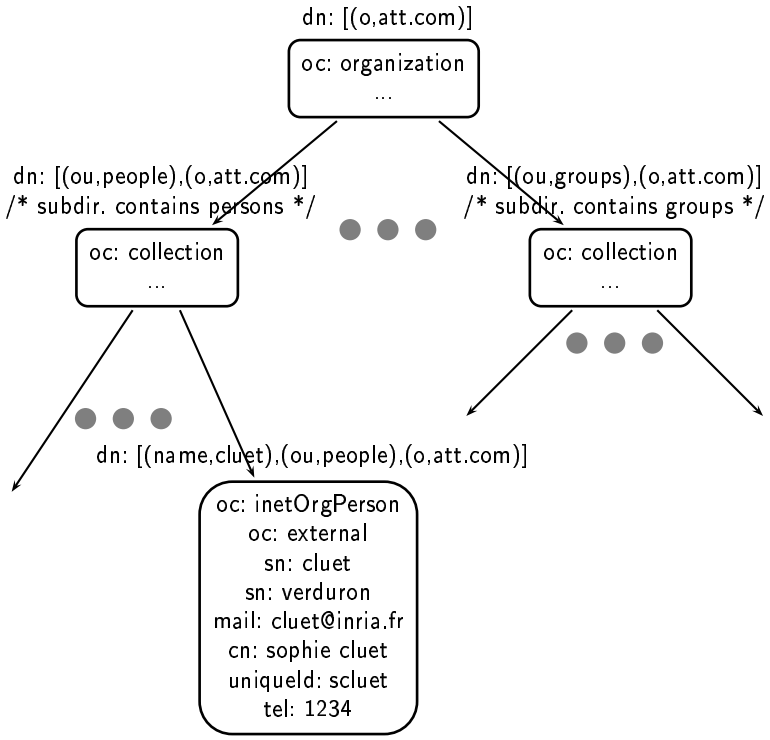

Fig. 1. LDAP instance example

Projection: Defines a set of attribute names that determines what data should be returned from each matching entry.

Example 1. Consider the query defined by the LDAP filter

$$([\text{(ou,people)}, \text{(o,att.com)}], \text{sub}, \{\&(\text{sn} = olga*)(\text{age} < 25)\}, \{\text{oc}, \text{sn}\})$$

would start the search at the [(ou,people), (o,att.com)] entry, test all its descendants against the filter expression, and return only the oc and sn attribute-value pairs.

2.2 Logical Framework

In order to build our logical framework, we first need to formalize the elements that make up the definition of the directory schema and the directory instance as far as they are relevant for our purposes. Based on these definitions we give a logical reformulation of an LDAP directory schema and instance. Later we shall introduce a set of additional axioms that capture the nature of an LDAP directory more comprehensively.

Definition 1 (Directory Schema). *A directory schema S is a 4-tuple $S = (\mathcal{C}, \mathcal{A}, \mathcal{D}, \rho)$, such that:*

- *$\mathcal{C}$ is a finite set of classes.*
- *$\mathcal{A}$ is a finite set of attributes such that $\{oc, dn\} \subseteq \mathcal{A}$.*
- *$\mathcal{D}$ (Domain) is an infinite ordered set containing all possible values an LDAP attribute can take. In particular, this set contains all class names, attribute names, and distinguished names.*
- *ρ is a pair of functions $\rho_1, \rho_2 : \mathcal{C} \to 2^{\mathcal{A}}$ that specify what attributes are needed for each class. ρ_1 defines the required attributes and ρ_2 the allowed attributes. Moreover, $\rho_1(c) \cap \rho_2(c) = \emptyset$ and $\{oc, dn\} \subseteq \rho_1(c)$, for all $c \in \mathcal{C}$.* □

Definition 2 (Directory Instance). *A directory instance I is a 4-tuple $I = (\mathcal{E}, \gamma, \delta, \alpha)$, such that:*

- *$\mathcal{E}$ is a finite set of directory entries.*
- *γ is a function of the form $\gamma : \mathcal{E} \to 2^{\mathcal{C}}$ that associates each entry with a non-empty set of classes from $\mathcal{C}$.*
- *δ assigns a unique distinguished name to each entry in $\mathcal{E}$.*
- *α is a function of the form $\alpha : \mathcal{E} \to 2^{\mathcal{A}' \times \mathcal{D}}$, $\mathcal{A}' = \mathcal{A} \setminus \{oc, dn\}$, that associates a set of attribute-value pairs to each entry.* □

Example 2. Consider the entry in Figure 1, we have:

α([(ou,people), (o,att.com)]) =
{(sn,cluet), (sn, verduron), (mail,cluet@inria.fr),
(cn, sophie cluet), (uniqueId,scluet), (tel, 1234)}

□

A logical reformulation of an LDAP directory schema and instance then can be stated as follows:

Definition 3 (LDAP Relations). *Let $S = (\mathcal{C}, \mathcal{A}, \mathcal{D}, \rho)$ be an LDAP schema and $I = (\mathcal{E}, \gamma, \delta, \alpha)$ a LDAP instance to S.*

- *An LDAP schema S is represented by a relation $\mathsf{schema}(\mathsf{class}, \mathsf{attr}, req)$, where $\mathsf{class} \in \mathcal{C}$, $\mathsf{attr} \in \mathcal{A}$, and req is a boolean value that indicates whether an attribute attr is required or allowed by class. An instance of schema is defined according to ρ: $\mathsf{schema}(c, a, \texttt{true})$ iff $a \in \rho_1(c)$, respectively, $\mathsf{schema}(c, a, \texttt{false})$ iff $a \in \rho_2(c)$, where $c \in \mathcal{C}$ and $a \in \mathcal{A}$.*
- *An LDAP instance I is represented by a relation $\mathsf{inst}(\mathsf{entry}, \mathsf{attr}, \mathsf{val})$, where $\mathsf{entry} \in \mathcal{E}$, $\mathsf{attr} \in \mathcal{A}$, and $\mathsf{val} \in \mathcal{D}$. Instances of inst are defined according to γ, δ and α. $(e, oc, v) \in \mathsf{inst}$ iff $v \in \gamma(e)$, $(e, dn, v) \in \mathsf{inst}$ iff $v = \delta(e)$, and $(e, a, v) \in \mathsf{inst}$ iff $(a, v) \in \alpha(e)$, where $e \in \mathcal{E}$, $a \in \mathcal{A}$, and $v \in \mathcal{D}$.* □

The unique schema information in LDAP makes the query processing and optimization a more difficult task than for conjunctive queries over a relational schema: there are several *hidden* rules implied by the schema definition which do not exist in the traditional relational model (e.g. the attribute of a class can be either `required`, or `allowed`). In Section 3 a set of integrity constraints originated from LDAP schema will be logically formalized and play an important role in semantic query optimization.

Multivalue attributes. As introduced above, when an entry has multi-valued attributes, then the transformation of the operator '!' is not trivial any more. Consider the example entry in Figure 1 where the attribute sn has two values cluet and verduron. To illustrate, let the filter be !(sn = cluet), then it is semantically not equal to the transformation $\mathsf{inst}(X, \mathsf{sn}, Z), Z \neq \mathsf{cluet}$. The first one will yield "false", whereas the second one returns the answer $X =$ [(name,cluet), (ou,people), (o,att.com)], $Z =$ verduron. Instead, we have to cope with the following cases: (i) the entry does not have sn as attribute, that is, the attribute sn is not defined; (ii) even if the entry has sn defined as attribute, but it has two values for it (like in our example), then the transformation will yield false results too. As a result, we need to define the additional predicates shown in Definition 4 and 5.

Taking into account that attributes may or may not be defined for certain entries, we are now going to introduce two further relations def and undef as follows:

Definition 4 (Definedness). *Let* $\mathsf{def}(\mathsf{entry}, \mathsf{attr})$ *and* $\mathsf{undef}(\mathsf{entry}, \mathsf{attr})$ *be relation schemata, where* entry *is of type* $\mathcal{E}$ *and* attr *is of type* $\mathcal{A}$*. Instances to* def *and* undef *are given by:*

- $\mathsf{def}(X, Y)$ *iff* $\exists Z : \mathsf{inst}(X, Y, Z)$.
- $\mathsf{undef}(X, Y)$ *iff* $\nexists Z : \mathsf{inst}(X, Y, Z)$.

In both cases the domain of quantification of the variables X, Y, Z *is* $\mathcal{E}$, $\mathcal{A}$, *respectively,* $\mathcal{D}$. □

Note, that according to this definition def and undef are guaranteed to be finite. We can understand undef as a *safe* negation of def, i.e., $\forall X \in \mathcal{E}, \forall Y \in \mathcal{A} : (\neg\mathsf{def}(X, Y) \equiv \mathsf{undef}(X, Y))$.

In order to give the negation operator '!' the correct semantics, we have to first introduce a set of predicates:

Definition 5 (Arithmetics). *Let* $\mathsf{eq}(\mathsf{entry}, \mathsf{attr}, \mathsf{val})$, $\mathsf{greater}(\mathsf{entry}, \mathsf{attr}, \mathsf{val})$, $\mathsf{less}(\mathsf{entry}, \mathsf{attr}, \mathsf{val})$, *and* $\mathsf{greater\_eq}(\mathsf{entry}, \mathsf{attr}, \mathsf{val})$, $\mathsf{less\_eq}(\mathsf{entry}, \mathsf{attr}, \mathsf{val})$ *be relations, where* $\mathsf{entry} \in \mathcal{E}$, $\mathsf{attr} \in \mathcal{A}$, *and* $\mathsf{val} \in \mathcal{D}$*. Instances of the relations are*

given by:

$$\begin{array}{ll} \mathit{eq}(X,Y,V) & \text{:- } \mathit{inst}(X,Y,Z), Z = V. \\ \mathit{greater}(X,Y,V) & \text{:- } \mathit{inst}(X,Y,Z), Z > V. \\ \mathit{less}(X,Y,V) & \text{:- } \mathit{inst}(X,Y,Z), Z < V. \\ \mathit{greater\_eq}(X,Y,V) & \text{:- } \mathit{inst}(X,Y,Z), Z \geq V. \\ \mathit{less\_eq}(X,Y,V) & \text{:- } \mathit{inst}(X,Y,Z), Z \leq V. \end{array}$$

(See also Example 5). □

Formulas in our framework are built over the relations inst, schema, def, undef, eq, greater, less, greater_eq, less_eq and arithmetic predicates as used in the LDAP filter language. It will suffice to restrict our logical language to formulas in DNF (Disjunctive Normal Form), as we shall see soon. Moreover, because of the filter syntax, all arithmetic predicates have the *semi-interval* property [13].

In the sequel, we denote an LDAP filter by F^* and its filter expression by F. Any LDAP filter expression F can be transformed into the logical model as follows:

In the following, X denotes a distinguished variable reused in any transformation step; in contrast, variable Z in any transformation step will represent a new variable, say $Z_1, Z_2, \ldots$, distinct from all variables previously introduced.

1. Replace '&' by '$\wedge$', '|' by '$\vee$', and '!' by '$\neg$', and transform the resulting formula into DNF.
2. Transform every positive occurrence of $(a = *)$ into $\mathsf{inst}(X, a, Z)$, every negative occurrence $!(a = *)$ to $\mathsf{undef}(X, a)$.
3. Consider now expressions different to $(a = *)$. Transform every positive occurrence of $(a\ op\ v)$ into the conjunction $(\mathsf{inst}(X, a, Z) \wedge (Z\ op\ v))$, and every negative occurrence $!(a\ op\ v)$ into the formula $\neg f(op)(X, a, v)$ [2].
4. Reestablish disjunctive normal form and call the resulting expression $\Sigma(F)$.

Example 3. Following the transformation algorithm, the filter

$$\&(\mathsf{uniqueId} = olga*)(!(\mathsf{mail} = *))$$

is transformed to

$$\mathsf{inst}(X, \mathsf{uniqueId}, Z) \wedge (Z = olga*) \wedge \mathsf{undef}(X, \mathsf{mail}).$$

In contrast, the filter

$$\&(\mathsf{uniqueId} = olga*)(!(\mathsf{mail} = olga*))$$

is transformed to

$$\mathsf{inst}(X, \mathsf{uniqueId}, Z_1) \wedge (Z_1 = olga*) \wedge \neg\mathsf{eq}(X, \mathsf{mail}, olga*).$$

□

[2] where $f(op)$ is the function of the five operators as the following: $f(=) = \mathsf{eq}$, $f(>) = \mathsf{greater}$, $f(<) = \mathsf{less}$, $f(\geq) = \mathsf{greater\_eq}$, and $f(\leq) = \mathsf{less\_eq}$.

Next the **Scope** and **Base** parts of the LDAP query are considered. We borrow the list expression syntax of Prolog to represent the DN, such that each DN of any entry is a list, in which each element is an attribute-value pair. As a result, we introduce two new predicates one and sub which have the meaning of child and descendant in XPath respectively.

Definition 6 (LDAP Scope Relation). *The semantics of* $\mathit{one}(dn1, dn2)$ *and* $\mathit{sub}(dn1, dn2)$ *are logically described as follows: where* $dn1 \in \mathcal{E}$ *and* $dn2 \in \mathcal{E}$*:*

- $\mathit{one}(Parent, [RDN|Parent])$. *(fact)*
- $\mathit{one}(X, Y) \rightarrow \mathit{sub}(X, Y)$. *(base)*
  $\mathit{one}(X, Y), \mathit{sub}(Y, Z) \rightarrow \mathit{sub}(X, Z)$. *(trans)*

Together with `fact`*, the rule* `trans` *can be rewritten to the following constraint which means that if* Z *is the descendant of one node, then it is also the descendant of the parent of that node:*

- $\mathit{sub}([RDN|Parent], Z) \rightarrow \mathit{sub}(Parent, Z)$. *(trans')*

□

Note that the rules `fact` and `trans'` have the form of *inclusion dependencies*. In [25], a generalized form of *inclusion dependencies* called *referential constraints* is introduced and the query containment problem in the presence of such constraints is studied. In the later part of the paper, we will give an example to show how the *referential constraints* affect the LDAP query containment checking.

The transformation of the **Base** and **Scope** is simple. The additional predicate sub(**Base**, X) for the scope sub and one(**Base**, X) for one is added to the filter expression F respectively. Note that the variable X is the same one as in the filter expression, and **Base** is always the DN of the given root entry. As for the scope base, one arithmetic predicate ($X =$ **Base**) is needed to be added in F, where X must also be the same variable as that in F.

The complete transformation.

$\Sigma(F)$ is the disjunction of the form $\{G_1, \ldots, G_k\}(k \geq 1)$ and each $G_i(1 \leq i \leq k)$ has the form:

$$R_1(\bar{Y_1}) \wedge ... \wedge R_n(\bar{Y_n}) \wedge \neg P_1(\bar{W_1}) \wedge ... \wedge \neg P_m(\bar{W_m}) \wedge C_Q$$

in which $R_1, ..., R_n(n \geq 1)$ are taken from the relations {inst, def, undef, sub, one}; $P_1, ..., P_m(m \geq 0)$, are predicates as given in Definition 5; and C_Q is composed of predicate expressions $u\theta v$, where $\theta \in \{=, \neq, <, >, \leq, \geq\}$, u is the element of $\bar{Y_1}, ..., \bar{Y_n}$, and v is a constant. Furthermore, for the sake of *safe negation*, we have $\{\bar{W_1}, ..., \bar{W_m}\} \subseteq \{\bar{Y_1}, ..., \bar{Y_n}\}$, to ensure that all the variables in negated predicates appear in the positive ones.

Now the projection part of an LDAP filter has to be taken into account. In the logical model, any projected attribute gives rise to a conjunctive query. In particular, because $\Sigma(F)$ can be considered a set of conjunctions, for each projected attribute a we get a set $Q_a = \{Q_{a1}, \ldots, Q_{ak}\}$ of conjunctive queries, each Q_{a_i}, $1 \leq i \leq k$, has the form

$$q_{a_i}(X, a, Z) \text{ :- } \mathsf{inst}(X, a, Z) \wedge G_i.$$

For each conjunctive query the variable Z is distinct from X and from all variables in the respective G_i, and is used to *copy* the value of the projected attribute into the query result.[3] The domain of quantification of variable X is the set of entries $\mathcal{E}$, where for the Y's and Z the domain is the set of possible values. To simplify notation, quantifiers are omitted and '$\wedge$' is replaced by comma. The set of conjunctive queries corresponding to an LDAP filter F^* is denoted by $\Sigma(F^*)$.

Up to now, we have completely transformed LDAP into the logical formalization. The final transformation is demonstrated by the following example:

Example 4. Again considering the Example 1, the LDAP query is transformed into two conjunctive queries since two attributes oc and sn should be projected. We have then two queries with oc and sn at the head respectively.

$$\begin{aligned} q1(X, \mathsf{oc}, Z1) \text{ :- } & \mathsf{sub}([(\mathsf{ou},\mathsf{people}),(\mathsf{o},\mathsf{att.com})], X), \mathsf{inst}(X, \mathsf{oc}, Z1), \mathsf{inst}(X, \mathsf{sn}, Z2), \\ & \mathsf{inst}(X, \mathsf{age}, Z3), Z2 = olga*, Z3 < 25. \\ q2(X, \mathsf{sn}, Z2) \text{ :- } & \mathsf{sub}([(\mathsf{ou},\mathsf{people}),(\mathsf{o},\mathsf{att.com})], X), \mathsf{inst}(X, \mathsf{sn}, Z2), \\ & \mathsf{inst}(X, \mathsf{age}, Z3), Z2 = olga*, Z3 < 25. \end{aligned}$$

□

Theorem 1. *Without considering the LDAP schema, the LDAP query can be equivalently transformed to Datalog with arithmetic comparison and stratified negation.* □

2.3 Containment of LDAP Queries

As long as the transformation of LDAP queries to *Datalog* is complete, we can conclude that the containment problem of LDAP can be reduced to an equivalent containment problem of *Datalog*, which is unfortunately undecidable [19]. However, the only recursions which cause the undecidability are the transitive closure rules, namely `base` and `trans'` in Definition 6. We argue that since we regard the definitions by *referential constraints* on the one and sub relations, instead of rules which *define* these IDB relations, the transformed query is *non-recursive Datalog* together with *referential constraints* for which query containment is decidable.

Theorem 2. *Containment of basic LDAP queries is decidable, and it is polynomially equivalent to the containment problem of non-recursive Datalog.* □

[3] If the literal $\mathsf{inst}(X, a, Z)$ is already contained in G_i, it is omitted.

The general query containment problem in *Datalog* has been extensively studied [19, 6]. However, several interesting cases originating from the LDAP query model are worth of discussing here.

Containment of queries with stratified negation. The containment of Datalog with stratified negation is studied in [14], however, the algorithm provided in [14] tests *uniform equivalence* of *Datalog* queries and actually, the algorithm would fail to prove in the above example that $Q_1 \subseteq Q_2$, which is obviously true. In [24], an algorithm is given to check the containment for (i) conjunctive queries with safe negation, and (ii) non-recursive Datalog with stratified negation which is the case here.

Example 5. Let the filters of two LDAP queries be the following:

$f_1 : \&((\mathsf{sn} = *)\ !(\mathsf{tel} = *))$,

$f_2 : \&((\mathsf{sn} = *)\ !(\mathsf{tel} = 1234*))$.

Let both queries have sn as projected attribute.

Q_1 and Q_2 can then be built up as the following (to simplify the queries, we assume that the **Scope** and **Base** of the queries are identical, so that they are omitted here):

$$Q_1 : q(X, \mathsf{sn}, Z) :- \mathsf{inst}(X, \mathsf{sn}, Z), \mathsf{undef}(X, \mathsf{tel}).$$
$$Q_2 : q(X, \mathsf{sn}, Z) :- \mathsf{inst}(X, \mathsf{sn}, Z), \neg\mathsf{eq}(X, \mathsf{tel}, 1234*).$$

Note that according to Definition 5, $\mathsf{eq}(X, \mathsf{tel}, 1234*)$ holds if X has *any* attribute value for tel which matches $1234*$. □

Containment of queries with scope constraints. As we noticed in the LDAP scope definition, the *referential constraints* are introduced to represent the transitive closure of the sub relation (see the rules `base` and `trans'` in Definition 6). As a result, the containment problem in the presence of *integrity constraints* must be considered. In [25], an algorithm for checking conjunctive query containment together with *implication constraints* and *referential constraints* is given. Due to the size limit of the paper, we only state it informally as the following: if the containment of queries Q_1 and Q_2 ($Q_1 \subseteq Q_2$) is to be checked, we first *expand* the query Q_1 by *referential expansion* using the *referential constraints* according to [25], and then the normal containment checking is executed.

The following example shows how the algorithm is applied in the LDAP logic model.

Example 6. Consider the following LDAP queries LQ_1 and LQ_2:

$$LQ_1 : ([\mathsf{ou{=}people,o{=}att.com}], \mathsf{one}, \{(\mathsf{sn} = *)\}, \{\mathsf{sn}\})$$
$$LQ_2 : ([\mathsf{o{=}att.com}], \mathsf{sub}, \{(\mathsf{sn} = *)\}, \{\mathsf{sn}\})$$

The queries are rewritten as the following Q_1 and Q_2

$$Q_1 : q(X, \mathsf{sn}, Z) \text{ :- } \mathsf{one}([(\mathsf{ou,people}),(\mathsf{o,att.com})], X), \mathsf{inst}(X, \mathsf{sn}, Z).$$
$$Q_2 : q(X, \mathsf{sn}, Z) \text{ :- } \mathsf{sub}([(\mathsf{o,att.com})], X), \mathsf{inst}(X, \mathsf{sn}, Z).$$

Query Q_1 asks for entries with attribute sn and are children of the starting entry [(ou,people),(o,att.com)]; while Q_2 asks for entries which are decendants of the starting entry [(o,att.com)]. From the transitive closure it is easy to see that the answer set of Q_1 is contained in that of Q_2 since the scope of Q_1 is a *subtree* of that of Q_2.

Formally described, we first expand the atom $\mathsf{one}([\mathsf{ou},\mathsf{people}),(\mathsf{o.att.com})], X)$ in the body of Q_1. Due to the referential constraints `base` and `trans'` above, the expanded conjunction:

$$\{\mathsf{one}([\mathsf{ou},\mathsf{people}),(\mathsf{o.att.com})], X), \mathsf{sub}([\mathsf{ou},\mathsf{people}),(\mathsf{o.att.com})], X), \mathsf{sub}([(\mathsf{o.att.com})], X)\}.$$

is semantically equivalent to the original atom above. Then, it is not hard to obtain that $Q_1 \subseteq Q_2$ – in the presence of the *referential constraints*. □

3 Schema Integrity Constraints in LDAP

The LDAP data model can represent heterogeneity of entities in a very natural and flexible manner, however, the gain of such flexibility does not come for free. In order to capture the semantics of the LDAP schema, we need to formalize them as integrity constraints, namely *disjunctive referential constraints* and *implication constraints* which will be introduced below. Integrity constraints in traditional databases are used for semantic query optimization, cooperative query answering, and view updating, etc [8]. In the rest of the section, we first describe the *disjunctive referential constraints* and *implication constraints* in the case of LDAP model, then the algorithm of containment checking in the presence of such integrity constraints is given.

Definition 7 (Constraints in LDAP).

Disjunctive Referential Constraint 1: *If an entry has values for attribute Attr, then it must belong to at least one class for which Attr is an attribute:*

$$\mathit{inst}(X, Attr, Val) \rightarrow \bigvee_{c \in \mathcal{C}, \mathit{schema}(c, Attr, \_)} \mathit{inst}(X, \mathsf{oc}, c).$$

Disjunctive Referential Constraint 2: *Each entry must belong to at least one class:*

$$\mathit{undef}(X, Attr) \rightarrow \bigvee_{c \in \mathcal{C}} \mathit{inst}(X, \mathsf{oc}, c).$$

Implication Constraint: *An entry must have at least one value for each of the required attributes of its classes:*

$$\mathit{inst}(X, \mathsf{oc}, C) \wedge \mathit{schema}(C, Attr, \mathtt{true}) \wedge \mathit{undef}(X, Attr) \rightarrow .$$

Note (Syntax): an implication constraint is a formula of the form:

$$\forall(\bar{Y_1}, ...\bar{Y_m})\ [r_1(\bar{Y_1}), ..., r_m(\bar{Y_m}), C_Q \rightarrow \texttt{false}].$$

or simply

$$r_1(\bar{Y_1}), ..., r_m(\bar{Y_m}), C_Q \rightarrow .$$

where the predicates of the body are the same as that of the conjunctive query. The only difference is that the head of every implication constraint is either `false` *or simply empty. They describe the unsatisfiability of queries. In [8], this syntax is used as the generalized form of integrity constraints.*

□

Note that *disjunctive referential constraints* are a generalized form of *referential constraints* in [25], *integrity constraints* formalized in [8], and *embedded dependencies* in [3].

The general form of disjunctive referential constraints (DRC) is as follows:

$$\forall(\bar{Y_1}, ..., \bar{Y_m})\ [r_1(\bar{Y_1}), ..., r_m(\bar{Y_m}) \rightarrow \exists(Z_1, ..., Z_k)\ p_1(\bar{X_1}) \vee ... \vee p_n(\bar{X_n})].$$

where $p_1, ..., p_n (1 \leq n)$, and $r_1, ..., r_m (0 \leq m)$ are predicate names; $\bar{X_1}, ..., \bar{X_n}$, $\bar{Y_1}, ..., \bar{Y_m}$ are tuples of variables and constants; $\{Z_1, ..., Z_k\} = \{\bar{X_1}, ..., \bar{X_n}\} \setminus \{\bar{Y_1}, ..., \bar{Y_m}\}$. Note that if $m = n = 1$, the constraint will be reduced to a *referential constraint* as described in [25].

We show in [23] that instead of the standard referential expansion as described in [25], we need a *disjunctive referential expansion* to expand the query to a set of sub-queries which is semantically *equivalent* to the original query but includes the schema information. At the same time, each expanded sub-query is guaranteed to be *minimal* in the sense that it is not contained in any other sub-query in the set. The formal proof can be found in [23], we give here an example to illustrate how the integrity constraints affect the containment checking of queries in LDAP.

Example 7. Given the queries Q_1 and Q_2 where Q_1 asks the value of uniqueId of each entry that has no attribute cn, Q_2 asks also the value of uniqueId of all entries which are instances of the class external.

$Q_1 : q(X, \mathsf{uniqueId}, Z)$:- $\mathsf{sub}([\mathsf{o,att.com}], X), \mathsf{inst}(X, \mathsf{uniqueId}, Z), \mathsf{undef}(X, \mathsf{cn})$.
$Q_2 : q(X, \mathsf{uniqueId}, Z)$:- $\mathsf{sub}([\mathsf{o,att.com}], X), \mathsf{inst}(X, \mathsf{uniqueId}, Z), \mathsf{inst}(X, \mathsf{oc}, \mathsf{external})$.

It is not hard to see that without the integrity constraints, Q_1 is not contained in Q_2. However, if we first expand Q_1 using DRC1 above (in conjunction with the schema given in Table 1), the two sub-queries are obtained as follows:

$Q_1' : q(X, \mathsf{uniqueId}, Z)$:- $\mathsf{sub}([\mathsf{o,att.com}], X), \mathsf{inst}(X, \mathsf{uniqueId}, Z), \mathsf{undef}(X, \mathsf{cn}), \mathsf{inst}(X, \mathsf{oc,external})$.
$Q_1'' : q(X, \mathsf{uniqueId}, Z)$:- $\mathsf{sub}([\mathsf{o,att.com}], X), \mathsf{inst}(X, \mathsf{uniqueId}, Z), \mathsf{undef}(X, \mathsf{cn}), \mathsf{inst}(X, \mathsf{oc,internal})$.

Note that semantically, we have $Q_1 =_{DRC} Q_1' \cup Q_1''$, which means: any entry satisfies the query Q_1 must be the instance of either external or internal, or both. Consequently, we can test that $Q_1' \subseteq Q_2$ and $Q_1'' \subseteq Q_2$ in the presence of the *implication constraint* given in Definition 7. The latter case should be explained: in Q_1'' the node should not have cn as attribute but it is an instance of internal which has cn as *required* attribute – which is a contradiction. Thus Q_1'' is unsatisfiable.

As a result, we get $Q_1 \subseteq Q_2$ in the presence of the *integrity constraints*, which could not hold otherwise. □

Theorem 3. *The containment problem of LDAP queries with schema constraints is decidable.* □

As shown in [25], the complexity of testing query containment of conjunctive queries in the presence of *referential constraints* and *implication constraints* is polynomially equivalent to that of conjunctive queries, which is $\prod_2^p$-complete (with arithmetic comparison) [21].

4 Related Work

Semi-structured data models like XML have been intensively studied recently [2]. It is more difficult to give a relational semantics to the query language XPath, since its path expressions are not first-order expressible [1]. However, there have been several works on the relational semantics of the query languages (mostly a fragment of) XPath [1], XML-QL [17], and XQuery [20]. The early semi-structured data model SML can be transformed to *Datalog* with function symbols allowed [16]. Compared with the semi-structured data model, LDAP data model deploys the elegant *distinguished name* (DN) to express the path expression of entries of LDAP while at the same time keeping the first-order semantics.

The method of adding semantics of class schema constraints to the query processing first appears in [5] for the object-oriented data model. Disjunctions are applied to a single query to generate a set of *terminal queries* with class information for each object variable in the body of the query. However, the algorithm is more pragmatic and can be applied only for the object-oriented data model. Our method is the natural extension of the generalized integrity constraints and therefore can be applied more broadly.

In [1] disjunction (disjunctive embedded dependencies) is applied to the integrity constraints in the relational transformation of XPath, However, an extension of the traditional chase algorithm is applied which is different from our logical method.

[7] is the first paper concerning the query rewriting problem in LDAP. However, the data model and query model of LDAP are not built upon first-order

logic so that algorithms are pragmatic. We notice that query containment checking is the crucial point in dealing with query rewriting [9]. We believe that the results of this paper will be helpful in tackling the query rewriting problems in LDAP.

5 Conclusion

With the growing popularity of directories services in the World Wide Web, LDAP directory enabled networking is being promoted by companies including AT&T, Cisco, and IBM. Our focus on this paper, has been to present a first-order logic semantics of the LDAP query language. A complete transformation of LDAP queries to *Datalog* with generalized *integrity constraints* is given and the complexity of query containment of LDAP query model is analyzed with and without *integrity constraints* respectively.

Though independent of the implementation system, our results can be applied into any LDAP directory enabled systems. Actually, due to many benefits from using relational databases as storage systems for LDAP data, there have been implementations of LDAP that use the IBM DB2 database [18]. Our transformation of LDAP data model and query model provides the possibilities to cope with relational databases and LDAP enabled directories in a seamless platform in the data integration system.

The *integrity constraints* we analyzed in this paper can be used for query processing, semantic query optimization or semantic caching in LDAP systems. Furthermore, the problems of rewriting query using views which arise recently in the data integration system, can also substantially benefit from our results.

References

1. A. Deutsch and V. Tannen. Containment and Integrity Constraints for XPath Fragments. In *KRDB*, 2001.
2. S. Abiteboul, P. Buneman, and D. Suciu. Data on the web. Morgan Kaufmann, 2000.
3. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley Publishing Company, 1995.
4. S. Amer-Yahia, H. Jagadish, L. Lakshmanan, and D. Srivastava. On Bounding-schemas for LDAP Directories. Tech. Report, Concordia University, 1999.
5. E. P. F. Chan. Containment and Minimization of Positive Conjunctive Queries in OODB's. In *Proceedings of the Eleventh ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS).* ACM Press, 1992.
6. S. Chaudhuri and M. Y. Vardi. On the Equivalence of Recursive and Nonrecursive Datalog Programs. In *Proceedings of the Eleventh ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems.* ACM Press, 1992.
7. S. Cluet, O. Kapitskaia, and D. Srivastava. Using LDAP Directory Caches. In *Proceedings of the Eighteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems.* ACM Press, 1999.

8. P. Godfrey, J. Grant, J. Gryz, and J. Minker. Integrity constraints: Semantics and applications. In Jan Chomicki and Gunter Saake, editors, *Logics for Databases and Information Systems*. Kluwer, 1998.
9. A. Halevy. Answering Queries Using Views: A Survey. To appear in VLDB Journal, 2001.
10. T. Howes, M. Smith, and G. Good. Understanding and Deploying LDAP Directory Services. Macmillan Technical Publishing, 1999.
11. Innosoft. Innosoft's Resources for Directory Deployments. "`http://www.innosoft.com/ldap_survey`".
12. H. V. Jagadish, Laks V. S. Lakshmanan, Tova Milo, Divesh Srivastava, and Dimitra Vista. Querying Network Directories. In *SIGMOD*. ACM Press, 1999.
13. A. Klug. On conjunctive queries containing inequalities. In J. ACM 35:1, pp. 146–160, 1988.
14. A. Y. Levy and Y. Sagiv. Queries Independent of Updates. In *19th International Conference on Very Large Data Bases*, 1993.
15. P. J. Marron and G. Lausen. On Processing XML in LDAP. In *Proceedings of the VLDB*, 2001.
16. Y. Papakonstantinou. Query processing in heterogeneous information sources. PhD thesis, Dept. of Computer Science, Stanford University, 1996.
17. J. Shanmugasundaram, K. Tufte, C. Zhang, G. He, D. J. DeWitt, and J. F. Naughton. Relational Databases for Querying XML Documents: Limitations and Opportunities. In *Proceedings of 25th International Conference on Very Large Data Bases*, 1999.
18. S. S. B. Shi, E. Stokes, D. Byrne, C. F. Corn, D. Bachmann, and T. Jones. An enterprise directory solution with DB2. *IBM Systems Journal*, 39(2):360–383, 2000.
19. O. Shmueli. Decidability and Expressiveness of Logic Queries. In *Proceedings of the Sixth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*. ACM, 1987.
20. I. Tatarinov, Zachary Ives, Alon Halevy, and Dan Weld. Updating XML. In *Proc. of ACM SIGMOD Conf. on Management of Data*, 2001.
21. R. van der Meyden. The Complexity of Querying Indefinite Information: Defined Relations. Ph.D thesis, Rutgers University, 1992.
22. M. Wahl, T. Howes, and S. Kille. Lightweight Directory Access Protocol (v3). Request for Comments 2251. `http://www3.innosoft.com/ldapworld/rfc2251.txt`, 1997.
23. F. Wei and G. Lausen. Conjuctive query containment in the presence of disjunctive integrity constraints. Submitted.
24. F. Wei and G. Lausen. Query Containment for Conjunctive Queries and Datalog with Negation. Technical Report. "`http://www.informatik.uni-freiburg.de/~fwei/paper/containment.ps`".
25. X. Zhang and Z. Meral Özsoyoglu. Implication and referential constraints: A new formal reasoning. *TKDE*, 9(6):894–910, 1997.