Proceedings of the International Workshop on

Logic in Databases (LID 2009)

Leopoldo Bertossi
Henning Christiansen
(Editors)

OCTOBER 2009 ROSKILDE UNIVERSITY COMPUTER SCIENCE RESEARCH REPORT #127

Computer Science
Roskilde University
P. O. Box 260
DK–4000 Roskilde
Denmark

Telephone: +45 4674 3839
Telefax: +45 4674 3072
Internet: http://www.ruc.dk/dat_en/
E-mail: datalogi@ruc.dk

ISSN 0109–9779

Research reports are available electronically from:

http://www.ruc.dk/dat_en/research/reports/

Proceedings of

LID 2009

International Workshop on

Logic in Databases

Roskilde University, Denmark, October 29, 2009

Edited by

Leopoldo Bertossi & Henning Christiansen

Preface

The International Workshop on Logic in Databases, LID 2009, is a forum for bringing together researchers and practitioners in academia and industry from around the world who are focusing on all logical aspects of data management. The present LID workshop series started with **LID 2008** in Rome as the confluence of three successful events series which had a strong overlap in interests.

- **LID'96**, an international workshop on Logic in Databases, from which LID 2008 and 2009 has derived its name; July 1-2, 1996, San Miniato, Italy.
- **LAAIC'05** and **LAAIC'06**, international workshops on Logical Aspects and Applications of Integrity Constraints; Copenhagen, Denmark, August 26, 2005, associated with DEXA 2005, and Krakow, Poland, September 8, 2006, associated with DEAX 2006.
- **IIDB'06**, an international workshop on Inconsistency and Incompleteness in Databases; March 26, 2006, Munich, Germany, associated with EDBT 2006.

Mathematical logic is inevitably coupled to databases in both practice and research. A database schema together with its integrity constraints is a logical specification of properties of the data, determining both internal storage structure and the semantics of data in relation to the real world. Databases are the only widespread application of information technology whose inherent logic and algebraic nature is apparent to developers at all levels.

Reasoning about databases is central for research in new database models as well as for the development of particular applications. The appearance of the relational model in the 1970ies was a giant step for the recognition of declarative specifications which, influenced by the even earlier use of logic in artificial intelligence and knowledge representation, lead to deductive databases and related models. This, in turn, has lead to powerful reasoning methods for relational databases as they can be considered as special cases of deductive databases.

This edition of LID has two invited presentations plus presentations of seven accepted papers that have been selected by an international programme committee. At request of the authors, one accepted paper is not included in the proceedings, but is presented at the workshop, *Sebastian Lehrack*, *Ingo Schmitt* and *Sascha Saretz*: CQQL: A Quantum Logic-Based Extension of the Relation Domain Calculus.

We are grateful to all people who contributed to LID 2009 in different ways, including all authors who submitted papers, to our invited speakers, Andrea Calì and Dan Olteanu, the programme and steering committees, reviewers, and to Roskilde University, the research group for Programming, Logic and Intelligent System and the CBIT department, for hosting the workshop.

October 2009
Leopoldo Bertossi & Henning Christiansen

Workshop Organization

Programme Chairs

Leopoldo Bertossi
Henning Christiansen

Programme Committee

Foto Afrati
Pablo Barceló
Alexander Borgida
Loreto Bravo
Marc Denecker
Wenfei Fan
Floris Geerts
Bart Kuijpers
Georg Lausen
Sebastian Link
Maarten Marx
Riccardo Rosati
Marie-Christine Rousset
Francesco Scarcello
Dan Suciu
Val Tannen
David Toman
Jef Wijsen
Peter Wood

External Reviewers

Matthew Damigos

Local Organization

Henning Christiansen

LID Steering Committee

Andrea Calì
Jan Chomicki
Henning Christiansen
Laks V.S. Lakshmanan
Davide Martinenghi
Dino Pedreschi
Jef Wijsen
Carlo Zaniolo

Workshop Website

http://lid2009.ruc.dk/

Copyright Notice

Table of Contents

Invited Contributions

Accepted Papers

Tractable Query Answering over Conceptual Schemata*

Andrea Calì[2,1], Georg Gottlob[1,2], and Andreas Pieris[1]

[1]Computing Laboratory, University of Oxford
[2]Oxford-Man Institute of Quantitative Finance, University of Oxford
{andrea.cali, georg.gottlob, andreas.pieris}@comlab.ox.ac.uk

Abstract. We address the problem of answering conjunctive queries (CQs) over extended Entity-Relationship schemata, which we call *EER (Extended ER)* schemata, with is-a among entities and relationships, and cardinality constraints. This is a common setting in conceptual data modelling, where reasoning over incomplete data with respect to a knowledge base is required. We adopt a semantics for EER schemata based on their relational representation. We identify a wide class of EER schemata for which query answering is tractable in data complexity; the crucial condition for tractability is the *separability* between maximum-cardinality constraints (represented as key constraints in relational form) and the other constraints. We provide, by means of a graph-based representation, a syntactic condition for separability: we show that our conditions is not only sufficient, but also necessary, thus precisely identifying the class of separable schemata. We show how tractable query answering (in AC_0 in data complexity) can be achieved by employing query rewriting techniques. We show that further negative constraints can be added to the EER formalism, while still keeping query answering tractable. We show that our formalism is general enough to properly generalise the most widely adopted knowledge representation languages.

1 Introduction

Since Chen's original Entity-Relationship formalism [15], conceptual modelling has been playing a prominent role in database design. More recently, logic-based formalisms have been employed for conceptual data modelling, in particular *Description Logics* [14]. Such formalisms have relevant applications especially in data exchange, information integration, semantic web, and web information systems, where the data, coming from different, heterogeneous sources, are in general incomplete/inconsistent w.r.t. constraints imposed by a conceptual schema. In such a setting, answering queries posed on the schema requires reasoning under a knowledge base constituted by the conceptual schema [6]. A relevant issue in query answering is tractability; in particular, what is commonly considered relevant here is the *data complexity* of query answering, i.e., the complexity in the

* This paper is a short version of [10].

case both the schema (plus, possibly, additional constraints) and the query are fixed, and the complexity is calculated considering the data as the only input parameter; this is natural, since the data size is normally much larger than the size of the schema and of the query. An important class of languages that guarantees tractable data complexity is the *DL-Lite* family [7, 23]. In particular, answering conjunctive queries (a.k.a. select-project-join queries) under DL-Lite knowledge bases is polynomial in data complexity; it is actually better than polynomial, more precisely, it is in AC_0 in data complexity, where AC_0 is the complexity of recognizing words in languages defined by constant-depth Boolean circuits with an (unlimited fan-in) AND and OR gates.

In this paper we consider an extended Entity-Relationship formalism, that we call EER[1], that comprises is-a among entities and relationships, mandatory and functional participation of entities to relationships, mandatory and functional attributes. The EER formalism is flexible and expressive, and at the same time well understood by database practitioners, differently from, for instance, Description Logics. We first illustrate, as in [6, 3, 4], a semantics of the EER formalism, by showing a translation of EER schemata into relational ones with a class of constraints (a.k.a. dependencies) called *conceptual dependencies (CDs)* [3]; in particular, CDs are *key dependencies (KDs)* and *tuple-generating dependencies (TGDs)* (more precisely, the TGDs in a set of CDs are *inclusion dependencies*). We then address the problem of answering conjunctive queries over EER schemata, that is, under CDs. Our contribution is the following. We identify a class of EER schemata, defined through a syntactic condition on the corresponding CDs, that guarantees *separation*, i.e., the absence of interaction between KDs and TGDs. We call such CDs *non-conflicting CDs (NCCDs)*. Answers to queries under NCCDs can be computed, if the data are consistent with the schema, considering TGDs only. The answers to queries posed on an EER schema represented with NCCDs, given an instance for that schema, can be computed by evaluating a *rewriting* of the original query on the original data, provided that the data are consistent with the schema. The existence of a query rewriting algorithm allows for tractable query answering. In particular, in our case, the computational complexity is AC_0 in data complexity (i.e., w.r.t. the data only). It is important to mention that the version of our algorithm that we present here is tailored for this particular, interesting case. The more general version [11] is capable of dealing with more expressive classes of constraints. We enrich the EER formalism by adding *negative constraints*, which serve to represent the fact that the data are inconsistent with respect to the schema, as well as further constraints enforcing, for example, (pairwise) disjointness between entities and relationships, and non-participation of an entity to a relationship. We show that adding negative constraints to CDs does not alter the computational complexity of conjunctive query answering.

The conceptual schemata for which we are able to compute query answering in a tractable way is general enough to comprise most practical cases.

[1] While we use the same name adopted in [21], our formalism is not the same as the one in this paper.

2 Preliminaries

2.1 Relational Model and Constraints, Queries, and Chase

We define the following pairwise disjoint (infinite) sets of symbols: *(i)* a set Γ of *constants*; constitute the "normal" domain of a database, and *(ii)* a set Γ_f of *labeled nulls*, used as placeholders for unknown values, and that can be also seen as variables. A lexicographic order is defined on Γ and Γ_f, such that every value in Γ_f follows all those in Γ. A *relational schema* $\mathcal{R}$ (or simply *schema*) is a set of *relational symbols* or *predicates*, each with its associated arity. We write r/n to denote that the predicate r has arity n. A *position* $r[i]$ (in a schema $\mathcal{R}$) is identified by a predicate $r \in \mathcal{R}$ and its i-th argument (or attribute). A *term* t is a constant, null, or variable. An *atomic formula* (or simply *atom*) has the form $r(t_1, \ldots, t_n)$, where r/n is a relation, and $t_1, \ldots, t_n$ are terms. For an atom α, we denote as $dom(\alpha)$ the set of terms occurring in α; this notation naturally extends to sets and conjunctions of atoms. A *relational instance* (or simply *instance*) D for a schema $\mathcal{R}$ is a (possibly infinite) set of atoms of the form $r(t)$ (a.k.a. *facts*), where $r/n \in \mathcal{R}$ and $t \in (\Gamma \cup \Gamma_f)^n$. We denote as $r(D)$ the set $\{t \mid r(t) \in D\}$. We will sometimes use the term *database* for a finite instance.

A *substitution* is a function $h : S_1 \to S_2$ defined as follows: *(i)* $\varnothing$ is a substitution (empty substitution); *(ii)* if h is a substitution then $h \cup \{X \to Y\}$ is a substitution, where $X \in S_1$ and $Y \in S_2$, and h does not already contain some $X \to Z$ with $Y \neq Z$. If $X \to Y \in h$ we write $h(X) = Y$. A *homomorphism* from a set of atoms A_1 to a set of atoms A_2, both over the same schema $\mathcal{R}$, is a substitution $h : dom(A_1) \to dom(A_2)$ such that: *(i)* if $t \in \Gamma$ then $h(t) = t$, and *(ii)* if $r(t_1, \ldots, t_n)$ is in A_1 then $h(r(t_1, \ldots, t_n)) = r(h(t_1), \ldots, h(t_n))$ is in A_2. If there are homomorhisms from A_1 to A_2 and vice-versa, we say that A_1 and A_2 are *homomorphically equivalent*.

A *conjunctive query* (CQ) q of arity n over a schema $\mathcal{R}$, written as q/n, is a formula of the form $q(\mathbf{X}) \leftarrow \varphi(\mathbf{X}, \mathbf{Y})$, where $\varphi(\mathbf{X}, \mathbf{Y})$ is a conjunction of atoms over $\mathcal{R}$, where $\mathbf{X}$ and $\mathbf{Y}$ are sequences of variables or constants in Γ, and $|\mathbf{X}| = n$. The atom $q(\mathbf{X})$ is the *head* of q, denoted as $head(q)$, and $\varphi(\mathbf{X}, \mathbf{Y})$ is the *body* of q, denoted as $body(q)$. A *union of conjunctive queries* (UCQ) of arity n over $\mathcal{R}$ is a set Q of CQs over $\mathcal{R}$, written as Q/n, where each $q \in Q$ has the same arity n, and uses the same symbol in the head. The *answer* to a CQ q/n of the form $q(\mathbf{X}) \leftarrow \varphi(\mathbf{X}, \mathbf{Y})$ over a database D, denoted as $q(D)$, is the set of all n-tuples $t \in \Gamma^n$ for which there exists a homomorphism $h : \mathbf{X} \cup \mathbf{Y} \to \Gamma \cup \Gamma_f$ such that $h(\varphi(\mathbf{X}, \mathbf{Y})) \subseteq D$, and $h(\mathbf{X}) = t$. The *answer* to a UCQ Q over D, denoted as $Q(D)$, is defined as the set $\{t \mid \exists\, q \in Q \text{ such that } t \in q(D)\}$.

Given a schema $\mathcal{R}$, a *tuple-generating dependency (TGD)* σ over $\mathcal{R}$ is a first-order formula of the form $\forall \mathbf{X} \forall \mathbf{Y}\, \varphi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z}\, \psi(\mathbf{X}, \mathbf{Z})$, where $\varphi(\mathbf{X}, \mathbf{Y})$ and $\psi(\mathbf{X}, \mathbf{Z})$ are conjunctions of atoms over $\mathcal{R}$, called the *body* and the *head* of σ, denoted as $body(\sigma)$ and $head(\sigma)$, respectively. Henceforth, to avoid notational clutter, we will omit the universal quantifiers in TGDs. A *key dependency (KD)* over $\mathcal{R}$ is an assertion of the form $key(r) = \mathbf{A}$, where $r \in \mathcal{R}$, and $\mathbf{A}$ is a set of attributes of r. A TGD of the form $\varphi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z}\, \psi(\mathbf{X}, \mathbf{Z})$ is satisfied by a

database D iff, whenever there exists a homomorphism h such that $h(\varphi(\mathbf{X}, \mathbf{Y})) \subseteq D$, there exists an extension h' of h (i.e., $h' \supseteq h$) such that $h'(\psi(\mathbf{X}, \mathbf{Z})) \subseteq D$. A KD of the form $key(r) = \mathbf{A}$ is satisfied by a database D iff, for each pair of distinct tuples $t_1, t_2 \in r(D)$, $t_1[\mathbf{A}] \neq t_2[\mathbf{A}]$, where $t[\mathbf{A}]$ is the projection of tuple t over $\mathbf{A}$.

We now define the notion of *query answering* under dependencies. Given a set Σ of dependencies over $\mathcal{R}$, and a database D for $\mathcal{R}$, the *models* of D w.r.t. Σ, denoted as $mods(D, \Sigma)$, is the set of all databases B such that B satisfies all the dependencies in Σ, and $B \supseteq D$. The *answer* to a CQ q w.r.t. Σ and D, denoted as $ans(q, \Sigma, D)$, is the set $\{t \mid t \in q(B) \text{ for each } B \in mods(D, \Sigma)\}$. The *decision problem* associated to query answering under dependencies is the following: given a set Σ of dependencies over $\mathcal{R}$, a database D for $\mathcal{R}$, a CQ q/n over $\mathcal{R}$, and an n-tuple $t \in \Gamma^n$, decide whether $t \in ans(q, \Sigma, D)$.

The *chase procedure* (or simply *chase*) is a fundamental algorithmic tool introduced for checking implication of dependencies [20], and later for checking query containment [18]. Informally, the chase is a process of repairing a database w.r.t. a set of dependencies so that the resulted database satisfies the dependencies. The chase works on an instance through the so-called TGD and KD *chase rules*. We shall use the term *chase* interchangeably for both the procedure and its result. The TGD chase rule comes in two different, equivalent fashions: *oblivious* and *restricted* [8], where the restricted one repairs TGDs only when they are not satisfied. In this paper we focus on the oblivious one for better technical clarity. The *chase* of a database D w.r.t. a set Σ_T of TGDs and a set Σ_K of KDs, denoted $chase(D, \Sigma)$, where $\Sigma = \Sigma_T \cup \Sigma_K$, is the (possibly infinite) instance constructed by iteratively applying *(i)* the TGD chase rule once, and *(ii)* the KD chase rule as long as it is applicable (i.e., until a fixpoint is reached). The chase rules follow.

TGD Chase Rule. Consider a database D for a schema $\mathcal{R}$, and a TGD $\sigma = \varphi(\mathbf{X}, \mathbf{Y}) \rightarrow \exists \mathbf{Z}\, \psi(\mathbf{X}, \mathbf{Z})$ over $\mathcal{R}$. If σ is *applicable* to D, i.e., there exists a homomorphism h such that $h(\varphi(\mathbf{X}, \mathbf{Y})) \subseteq D$ then: (i) define $h' \supseteq h$ such that $h'(Z_i) = z_i$ for each $Z_i \in \mathbf{Z}$, where $z_i \in \Gamma_f$ is a "fresh" labeled null not introduced before and following lexicographically all those introduced so far, and (ii) add to D the set of atoms in $h'(\psi(\mathbf{X}, \mathbf{Z}))$ if not already in D.

KD Chase Rule. Consider an instance D for a schema $\mathcal{R}$, and a KD η of the form $key(r) = \mathbf{A}$ over $\mathcal{R}$. If η is *applicable* to D, i.e., there are two (distinct) tuples $t_1, t_2 \in r(D)$ such that $t_1[\mathbf{A}] = t_2[\mathbf{A}]$, then for each attribute B of r s.t. $B \notin \mathbf{A}$: *(i)* if $t_1[B]$ and $t_2[B]$ are both constants of Γ, then there is a *hard violation* of η and the chase *fails*; in this case $mods(D, \Sigma) = \varnothing$ and we say that D is inconsistent with Σ; *(ii)* if $t_1[B]$ (resp., $t_2[B]$) is a constant of Γ and $t_2[B]$ (resp., $t_1[B]$) is a labeled null of Γ_f, then replace each occurrence of $t_2[B]$ (resp., $t_1[B]$) in D with $t_1[B]$ (resp., $t_2[B]$), and (iii) if $t_1[B]$ and $t_2[B]$ are both labeled nulls of Γ_f, then either replace each occurrence of $t_1[B]$ in D with $t_2[B]$ if the former follows lexicographically the latter, or vice-versa otherwise.

It is well-known that $chase(D, \Sigma)$ is a *universal instance* of D w.r.t. Σ, i.e., for each database $B \in mods(D, \Sigma)$, there exists a homomorphism from $chase(D, \Sigma)$

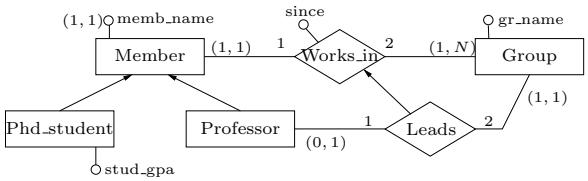


Fig. 1. EER Schema for Example 1.

to B [17]. Using this fact, it can be shown that the answers $ans(q, \Sigma, D)$ to CQ q/n under a set Σ of TGDs and KDs, in the case where the chase does not fail, can be obtained by evaluating q over $chase(D, \Sigma)$ (which is possibly infinite) and discarding tuples containing at least one null [17]. In case the chase fails, $ans(q, \Sigma, D)$ contains all tuples in Γ^n.

We say that a set Σ of constraints (not necessarily TGDs and KDs) is *first-order rewritable (or FO-rewritable)* [9, 23] iff, for every database D and for every CQ q, there exists a first-order query q_{FO} such that $q_{FO}(D) = ans(q, \Sigma, D)$.

2.2 The Conceptual Model

In this section we present the conceptual model we adopt in this paper, and we define it in terms of relational schemata with constraints. Our model incorporates the basic features of the ER model [15] and OO models, including subset (or is-a) constraints on both entities and relationships. We call our model *Extended Entity-Relationship (EER) model*. An *EER schema* consists of a collection of entity, relationship, and attribute definitions over an alphabet of symbols, partitioned into entity, relationship and attribute symbols. The model is similar as, e.g., the one in [3], and it can be summarised as follows: *(i)* entities and relationships can have attributes; an attribute can be mandatory (instances have at least one value for it), and functional (instances have at most one value for it); *(ii)* entities can participate in relationships; a participation of an entity E in a relationship R can be mandatory (instances of E participate at least once), and functional (instances of E participate at most once); *(iii)* is-a relations can hold between entities and between relationships. For further details see [3].

Example 1. The schema in Figure 1, based on the usual ER graphic notation, describes members of a university department working in research groups. The is-a constraints specify that Ph.D. students and professors are members, and that each professor works in the same group that (s)he leads. The cardinality constraint $(1, N)$ on the participation of *Group* in *Works_in*, for instance, specifies that each group has at least 1 member and no maximum number of members (symbol N). The participating entities to each relationship are numbered (each number identifies a *component*). □

The semantics of an EER schema $\mathcal{C}$ is defined by associating a relational schema $\mathcal{R}_\mathcal{C}$ to it, and then specifying when a database for $\mathcal{R}_\mathcal{C}$ satisfies all the

EER Construct	**Relational Constraint**
attribute A for an entity E	$a(X, Y) \rightarrow e(X)$
attribute A for a relationship R	$a(X_1, \ldots, X_n, Y) \rightarrow r(X_1, \ldots, X_n)$
rel. R with entity E as i-th component	$r(X_1, \ldots, X_n) \rightarrow e(X_i)$
mandatory attribute A of entity E	$e(X) \rightarrow \exists Y\, a(X, Y)$
mandatory attribute A of relationship R	$r(X_1, \ldots, X_n) \rightarrow \exists Y\, a(X_1, \ldots, X_n, Y)$
functional attribute A of an entity	$key(a) = \{1\}$ (a has arity 1)
functional attribute A of a relationship	$key(a) = \{1, \ldots, n\}$ (a has arity $n + 1$)
is-a between entities E_1 and E_2	$e_1(X) \rightarrow e_2(X)$
is-a between relationships R_1 and R_2	$r_1(X_1, \ldots, X_n) \rightarrow r_2(X_1, \ldots, X_n)$
mandatory part. of E in R (i-th comp.)	$e(X) \rightarrow r(X_1, \ldots, X_{i-1}, X, X_{i+1}, \ldots, X_n)$
functional part. of E in R (i-th comp.)	$key(r) = \{i\}$

Table 1. Derivation of relation constraints from an EER schema.

constraints imposed by the constructs of $\mathcal{C}$. We first define the relational schema that represents the so-called *concepts*, i.e., entities, relationships and attributes, of an EER schema $\mathcal{C}$ as follows: *(i)* each entity E in $\mathcal{C}$ has an associated predicate $e/1$; *(ii)* each attribute A of an entity E in $\mathcal{C}$ has an associated predicate $a/2$; *(iii)* each relationship R of arity n in $\mathcal{C}$ has an associated predicate r/n, and (*iv*) each attribute A of a relationship R of arity n in $\mathcal{C}$ has an associated predicate $a/(n+1)$. Intuitively, $e(c)$ asserts that c is an instance of entity E. $a(c, d)$ asserts that d is the value of attribute A (of some entity E) associated to c, where c is an instance of E. $r(c_1, \ldots, c_n)$ asserts that $(c_1, \ldots, c_n)$ is an instance of relationship R (among entities $E_1, \ldots, E_n$), where $c_1, \ldots, c_n$ are instances of $E_1, \ldots, E_n$, respectively. Finally, $a(c_1, \ldots, c_n, d)$ asserts that d is the value of attribute A (of some relationship R of arity n) associated to the instance $(c_1, \ldots, c_n)$ of R.

Queries are formulated using the relations in the relational schema we obtain from the EER schema as described above.

Example 2. Consider again the EER schema shown in Figure 1. The schema $\mathcal{R}_\mathcal{C}$ associated to $\mathcal{C}$ consists of *member*/1, *phd_student*/1, *professor*/1, *group*/1, *works_in*/2, *leads*/2, *memb_name*/2, *stud_gpa*/2, *memb_name*/2 and *since*/3. Suppose that we want to know the names of the students who work in the DB group since 2006. The corresponding CQ is

$$q(B) \leftarrow \mathit{phd\_student}(A), \mathit{memb\_name}(A, B), \mathit{works\_in}(A, C), \mathit{since}(A, C, 2006), \\ \mathit{memb\_name}(C, \mathit{db}).$$ □

We now define the semantics of the EER constructs. This is done by specifying, using the dependencies introduced in Section 2.1, what databases over $\mathcal{R}_\mathcal{C}$ satisfy the constraints imposed by the constructs of $\mathcal{C}$. We do that by making use of relational database dependencies, as shown in Table 1 (where we assume that the relationships are of arity n). Notice that, slightly differently from [3], we do not allow permutations of components in is-a between relationships; for example, we can never derive a TGD of the form $r_1(X_1, X_2, X_3) \rightarrow r_2(X_3, X_1, X_2)$. The dependencies we obtain are called *conceptual dependencies (CDs)* [3]. Observe

that the constraints in a set of CDs are *key* and *inclusion dependencies* [1], where the latter are a special case of TGDs.

3 Separability

In this section we introduce a novel class of CDs, namely, the *non-conflicting CDs (NCCDs)*. In a set of NCCDs, the TGDs and the KDs do not interact, so that answers to queries over an EER schema can be computed by considering the TGDs only, and ignoring the KDs, once it is known that the initial data are consistent with respect to the schema, i.e., the chase does not fail. This semantic property, whose definition is given below, is usually known as *separability* [9, 5]. Henceforth, when using the term TGD, we shall refer to TGDs that are part of a set of CDs (the results of this paper do not hold in case of general TGDs).

Definition 1. *Consider a set $\Sigma = \Sigma_T \cup \Sigma_K$ of CDs over a schema $\mathcal{R}$, where Σ_T are TGDs and Σ_K are KDs. Σ is said to be* separable *if for every instance D for $\mathcal{R}$, and for every CQ q/n, we have that either $chase(D, \Sigma)$ fails, or $ans(q, \Sigma, D) = ans(q, \Sigma_T, D)$.*

Before syntactically defining NCCDs, we need the notion of *CD-graph*.

Definition 2. *Consider a set Σ of CDs over a schema $\mathcal{R}$. The CD-graph for $\mathcal{R}$ and Σ is defined as follows: (i) the set of nodes is the set of positions in $\mathcal{R}$; (ii) if there is a TGD σ in Σ such that the same variable appears in a position p_b in the body and in a position p_h in the head, then there is an arc from p_b to p_h.*

A node corresponding to a position derived from an entity (resp., a relationship) is called an e-node (resp., an r-node). Moreover, an r-node corresponding to a position which is a unary key in a relationship is called a k-node. We are now ready to give the notion of NCCDs.

Definition 3. *Consider a set Σ of CDs over a schema $\mathcal{R}$, and let G be the CD-graph for $\mathcal{R}$ and Σ. Σ is said to be* non-conflicting *if the following condition is satisfied: for each path $v_1 \frown v_2 \frown \ldots \frown v_m$ in G, where $m \geqslant 3$, such that: (i) v_1 is an e-node, (ii) $v_2, \ldots, v_{m-1}$ are r-nodes, and (iii) v_m is a k-node, there exists a path in G of only r-nodes from v_m to v_2.*

Example 3. Let us consider the schema in Example 1, ignoring the attributes for simplicity. The CD-graph for the CDs associated to the EER schema are depicted in Fig. 2. The k-nodes are *works_in*[1], *leads*[1], and *leads*[2]. It is immediate to see that the CDs are NCCDs. □

The following example shows that the KD chase rule can be applied during the chase procedure with respect to a set of NCCDs.

Example 4. Let us consider the EER schema in Example 3, which we call $\mathcal{C}$. We omit for space reasons the CDs $\Sigma_{\mathcal{C}}$ associated to $\mathcal{C}$. Take $D = \{professor(p),$

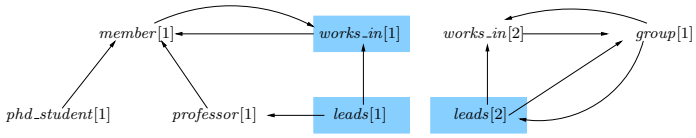


Fig. 2. CD-graph for Example 3. K-nodes are shaded.

$leads(p, g)\}$. In the computation of $chase(D, \Sigma_{\mathcal{C}})$, we add the atoms $member(p)$, $works\_in(p, g)$ and $works\_in(p, z_1)$, where $z_1 \in \Gamma_f$. Since $\Sigma_{\mathcal{C}}$ contains the KD $key(works\_in) = \{1\}$, we apply the KD chase rule and replace all occurrences of z_1 with g. □

We now establish the main result if this section, i.e., NCCDs are separable.

Theorem 1. *Consider a set $\Sigma = \Sigma_T \cup \Sigma_K$ of CDs over a schema $\mathcal{R}$, where Σ_T are TGDs and Σ_K are KDs. If Σ is non-conflicting, then it is separable.*

Proof (sketch). Let D be a database for $\mathcal{R}$ such that $chase(D, \Sigma)$ does not fail. By induction on the number of applications of the chase rule in the construction of $chase(D, \Sigma)$, it is possible to show that there exists homomorphism h such that $h(chase(D, \Sigma)) \subseteq chase(D, \Sigma_T)$. Moreover, since $chase(D, \Sigma) \in mods(D, \Sigma) \subseteq mods(D, \Sigma_T)$, and $chase(D, \Sigma_T)$ is a universal instance of D w.r.t. Σ_T, we get that there exists homomorphism μ such that $\mu(chase(D, \Sigma_T)) \subseteq chase(D, \Sigma)$. Therefore, $chase(D, \Sigma)$ and $chase(D, \Sigma_T)$ are homomorphically equivalent. The claim follows straightforwardly. □

We continue to show that the property of being non-conflicting is not only sufficient for separability, but also necessary. This way, we precisely characterise the class of separable EER schemata by means of a syntactic condition.

Theorem 2. *Consider a set $\Sigma = \Sigma_T \cup \Sigma_K$ of CDs over a schema $\mathcal{R}$, where Σ_T are TGDs and Σ_K are KDs. We have that if Σ is not non-conflicting, then it is not separable.*

Proof (sketch). We prove this result by exhibiting a database D for $\mathcal{R}$, and a Boolean[2] CQ q such that $chase(D, \Sigma)$ does not fail, and $\langle\rangle \in ans(q, \Sigma, D)$ but $\langle\rangle \notin ans(q, \Sigma_T, D)$. □

It is important to mention that results analogous to Theorems 1 and 2 hold for EER schemata with *binary* relationships only, and with is-a among relationships that allow for the *swapping* of the components (e.g., represented by a TGD of the form $r_1(X, Y) \rightarrow r_2(Y, X)$). The proofs, which we omit for space reasons, are analogous to those above. This result is important because with this variant of the EER formalism we are able to represent DL-Lite schemata.

Before moving to the next section, where we show that NCCDs are FO-rewritable, we prove here that CDs are in general not FO-rewritable.

[2] A Boolean CQ has no variables in the head, and has only the empty tuple $\langle\rangle$ as possible answer, in which case it is said that the query has *positive* answer.

Theorem 3. *General CDs are not FO-rewritable.*

Proof (sketch). We establish this result by giving a counterexample schema and a query such that no first-order rewriting exists for the query. □

4 Query Answering by Rewriting

In this section we address the problem of query answering under NCCDs by adopting query rewriting techniques. From the previous section, given a set Σ of CDs, once we know that the chase does not fail, we can concentrate only on the set Σ_T of TGDs that are in Σ. We adopt a query rewriting algorithm that allows us to answer CQs under TGDs by reformulating a given CQ q into a UCQ Q_r, that encodes the information about the given TGDs, and then evaluate Q_r over a given database to obtain the correct answers to q. The algorithm, which we omit here, is presented in detail in [10], and it can be considered as a variant of the rewriting algorithm in [12], which is designed to cope with inclusion dependencies. Our algorithm is an adapted and restricted version of a more general one, for which we refer the reader to [11], which is capable of dealing with a more general class of TGDs, with multiple atoms in the body.

5 Negative Constraints

In this section we show how the EER model can be extended, in the same fashion as in [9], with negative constraints.

A *negative constraint* on a schema $\mathcal{R}$ is a FO formula of the form $\forall \mathbf{X}\, \varphi(\mathbf{X}) \rightarrow \bot$, where $\varphi(\mathbf{X})$ is a conjunction of atoms over $\mathcal{R}$, and $\bot$ is the truth constant "false"; for simplicity, we will omit the universal quantifiers. Such a constraint is satisfied by a database D iff there is no homomorphism h such that $h(\varphi(\mathbf{X})) \subseteq D$.

We first show how to express the failure of the chase with negative constraints. Given an instance D, for each pair c_1, c_2 of *distinct* constants of Γ in $dom(D)$, we add to D the fact $neq(c_1, c_2)$, where neq is an auxiliary predicate. For every key constraint $key(r) = \{1, \ldots, m\}$, for a predicate r/n with $m < n$ (w.l.o.g., we assume the first m attributes to form the key; in particular, m can be only 1 or $n-1$), we add the following negative constraints, for all $j \in \{m+1, \ldots, n\}$:

$$r(X_1, \ldots, X_m, Y_{m+1}, \ldots, Y_n), r(X_1, \ldots, X_m, Z_{m+1}, \ldots, Z_n), neq(Y_j, Z_j) \rightarrow \bot.$$

As observed in [9], a constraint $\varphi(\mathbf{X}) \rightarrow \bot$ is satisfied by a database D iff the answer to the CQ $q() \leftarrow \varphi(\mathbf{X})$ over D is the empty set. Therefore, we can check the failure of the chase by answering such CQs, which has the same complexity as answering CQs under NCCDs. This implies FO-rewritability of NCCDs: we can answer a query q, given D and $\Sigma = \Sigma_T \cup \Sigma_K$, by evaluating over D the FO query obtained by taking the logical disjunction of the CQs associated to the negative constraints $\Sigma_\bot$, expressive the chase failure as above, and of the output of $\mathsf{rewrite}(\mathcal{R}, \Sigma_T, q)$ as in Section 4. We immediately get the following result.

Theorem 4. *Query answering on EER schemata represented by NCCDs is in* AC_0 *in data complexity.*

Negative constraints can be used to express several relevant constructs in EER schemata, for instance disjunction between entities and relationships, and non-participation of entities to relationships, but also more general ones.

Example 5. Consider an EER schema $\mathcal{C}$ obtained from the one in Example 1 (see Figure 1) by adding an entity *PensionScheme* and a relationship *Enrolled* between *PensionScheme* and *Member*, with no cardinality constraints; for space reasons, we do not show the new diagram. To express that students and professors are disjoint sets, we state $\mathit{phd\_student}(X), \mathit{professor}(X) \rightarrow \bot$ (entity disjunction). We can also express that a student cannot be enrolled in a pension scheme (i.e., it does not participate to *Enrolled*) with the negative constraint $\mathit{phd\_student}(X), \mathit{enrolled}(X, Y) \rightarrow \bot$ (non-participation). □

Consider a schema $\mathcal{R}$, a set of CDs Σ on $\mathcal{R}$, and a set of negative constraints $\Sigma_\bot$ on $\mathcal{R}$. The question remaining open so far is whether the fact that the CDs in Σ are NCCDs is necessary and sufficient to ensure separability of $\Sigma \cup \Sigma_\bot$. It is not difficult to show that for general negative constraints the property is not necessary; however, in particular cases it is. For example, we claim that if we restrict to negative constraints expressing entity and relationship disjunction plus non-participation, and to *strongly consistent* EER schemata [2], having NCCDs is necessary and sufficient for separability.

6 Discussion

Related work. The well-known Entity-Relationship model was introduced by the milestone paper of Chen [15]. A work giving a logic-based semantics is [16], which also provides an inference algorithm; [19] investigates cardinality constraints in the ER formalism. An investigation on reasoning tasks on different variants of ER schemata is found in [2]. Query answering is tightly related to query containment under constraints, a fundamental topic in database theory [13, 18, 3]. Data integration under ER schemata, strictly less expressive than EER schemata, is considered in [6]. [22] adopts a formalism which is more expressive than ours, thus not achieving similar tractability results. [13] considers query containment in a formalism similar to the EER model with less expressive negative constraints, focusing on decidability and combined complexity (i.e., the complexity w.r.t. the data, the schema and the query). No results on data complexity, nor a practical algorithm, are provided. A query rewriting algorithm for IDs and so-called *non-conflicting* KDs is presented in [12]. The works on DL-Lite [7, 23] exhibit tractable query answering algorithm (in AC_0 in data complexity) for different languages in the DL-Lite family. Recent works [8, 9] deal with expressive rules (TGDs) that consitute the languages of the *Datalog*$^\pm$ family, which are capable of capturing the EER formalism presented here, if we consider TGDs only. The languages in the *Datalog*$^\pm$ family are more expressive (and less tractable) than ours except for *Linear Datalog*$^\pm$, that allows

for query answering in AC_0 in data complexity. However, the class of NCCDs is not expressible in Linear Datalog$^{\pm}$ (plus the class of KDs presented in [9]), and moreover the FO-rewriting algorithm in [9], unlike ours, is not very well-suited for practical implementations. Finally, the works [3, 4] deal with general (not non-conflicting) CDs: PTIME data complexity of answering is obtained by paying a high price in combined complexity.

Conclusions and future work. In this paper we have identified, by means of a graph-based representation, a class of extended Entity-Relationship schemata for which query answering is tractable, and more precisely in AC_0 in data complexity. The tractability of answering in our setting hinges on the notion of separability, for which we have provided a precise characterisation in terms of a necessary and sufficient syntactic condition. We have shown that it is possible to answer queries on EER schemata, by means of a query rewriting. Our algorithm is an adapted version of a more general algorithm which can deal with much more expressive TGDs, and it can be considered to be a variant of the one in [12]. We have also shown that negative constraints can be added to EER schemata, without increasing the data complexity of query answering. The class of EER schemata we deal with is general enough to include most conceptual modelling and knowledge representation formalisms; in particular, it is strictly more expressive than the languages in the DL-Lite family.

We plan to extend our results by studying the combined complexity of query answering problem under NCCDs, and employing variants of our general rewriting algorithm to deal with even more expressive constraints. It is also our intention to run experiments with the techniques presented here.

Acknowledgments. The authors acknowledge support by the EPSRC project "Schema Mappings and Automated Services for Data Integration and Exchange" (EP/E010865/1). Georg Gottlob's work was also supported by a Royal Society Wolfson Research Merit Award.

References

1. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
2. A. Artale, D. Calvanese, R. Kontchakov, V. Ryzhikov, and M. Zakharyaschev. Reasoning over extended ER models. In *Proc. ER 2007*, pp. 277–292, 2007.
3. A. Calì. Containment of conjunctive queries over conceptual schemata. In *Proc. of DASFAA 2006*, pp. 628–643, 2006.
4. A. Calì. Querying incomplete data with logic programs: ER strikes back. In *Proc. of ER 2007*, pp. 245–260, 2007.
5. A. Calì, D. Lembo, and R. Rosati. On the decidability and complexity of query answering over incosistent and incomplete databases. In *Proc. of PODS 2003*, pp. 260–271, 2003.
6. A. Calì, D. Calvanese, G. De Giacomo, and M. Lenzerini. Accessing data integration systems through conceptual schemas. In *Proc. of ER 2001*, pp. 270–284, 2001.
7. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable reasoning and efficient query answering in description logics: the DL-Lite family. *J. Autom. Reasoning*, 39(3):385–429, 2007.

8. A. Calì, G. Gottlob, and M. Kifer. Taming the infinite chase: query answering under expressive relational constraints. In *Proc. of KR 2008*, pp. 70–80, 2008. Revised version available at `http://benner.dbai.tuwien.ac.at/staff/gottlob/CGK.pdf`.
9. A. Calì, G. Gottlob, and T. Lukasiewicz. A general datalog-based framework for tractable query answering over ontologies. In *Proc. of PODS 2009*, to appear, 2009.
10. A. Calì, G. Gottlob, and A. Pieris. Tractable query answering over conceptual schemata. *Proc. of ER 2009*, 2009. To appear.
11. A. Calì, G. Gottlob, and A. Pieris. Tractable query answering over conceptual schemata. Unpublished technical report, available from the authors, 2009.
12. A. Calì, D. Lembo, and R. Rosati. Query rewriting and answering under constraints in data integration systems. In *Proc. of IJCAI 2003*, pp. 16–21, 2003.
13. D. Calvanese, G. De Giacomo, and M. Lenzerini. On the decidability of query containment under constraints. In *Proc. PODS 1998*, pp. 149–158, 1998.
14. D. Calvanese, M. Lenzerini, and D. Nardi. Description logics for conceptual data modeling. Logics for Databases and Information Systems, pp. 229–263, 1998.
15. P. P. Chen. The entity-relationship model: towards a unified view of data. *ACM TODS*, 1(1):124–131, 1995.
16. G. Di Battista, and M. Lenzerini. A deductive method for entity-relationship modeling. In *Proc. of VLDB 1989*, pp. 13–21, 1989.
17. R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: semantics and query answering. *TCS*, 336(1):89-124, 2005.
18. D. S. Johnson, and A. C. Klug. Testing containment of conjunctive queries under functional and inclusion dependencies. *JCSS*, 28(1):167–189, 1984.
19. M. Lenzerini, and G. Santucci. Cardinality constraints in the entity-relationship model. In *Proc. of ER 1983*, pp. 529–549, 1983.
20. D. Maier, A. O. Mendelzon, and Y. Sagiv. Testing implications of data dependencies. *ACM TODS*, 4(4):455–469, 1979.
21. V. M. Markowitz, and J. A. Makowsky. Identifying extended entity-relationship object structures in relational schemas. *IEEE Trans. Software Eng.*, 16(8):777–790, 1990.
22. M. Ortiz, D. Calvanese, and T. Eiter. Characterizing data complexity for conjunctive query answering in expressive description logics. In *Proc. of AAAI 2006*, pp. 2006.
23. A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati. Linking data to ontologies. *J. Data Semantics*, 10:133–173, 2008.

A Toolbox of Query Evaluation Techniques for Probabilistic Databases

Dan Olteanu

Oxford University Computing Laboratory

We study the problem of query evaluation in probabilistic databases and survey some of the most promising existing techniques recently proposed by the database community. Although this problem is subsumed by general probabilistic inference, we argue that two fundamental aspects of databases, that is, (i) the separation of (very large) data and (small and fixed) query, and (ii) the use of mature relational query engines, can lead to more scalable techniques.

We survey both exact and approximate query evaluation techniques. In case of exact evaluation, we discuss syntactical restrictions of the language of conjunctive queries with inequalities, under which the queries become tractable [2, 6] (in general, the problem is #P-hard [1]). For these queries, we also show how relational query plans extended with efficient aggregation operators can be successfully used to evaluate them [1, 7]. At their core, these aggregations compute and combine, in a few scans over the query answers, probabilities of fragments of binary decision diagrams encoding the uncertainty in the query answer [5]. In case of intractable queries, we sketch an exact technique that first decomposes the data-query instance into a tractable subinstance, which is solved as before, and a (usually much smaller) intractable subinstance that can be solved either approximately or by employing AI inference techniques based on graph-theoretic properties of the instance, such as treewidth [3]. Alternatively, intractable queries can be evaluated using (deterministic or randomized) approximation techniques with error guarantees [4, 8].

References

1. Dalvi and Suciu. "Efficient Query Evaluation on Probabilistic Databases". In *VLDBJ*, 2007.
2. Dalvi and Suciu. "The Dichotomy of Conjunctive Queries on Probabilistic Structures". In *PODS*, 2007.
3. Jha, Olteanu, and Suciu. "Bridging the Gap Between Intensional and Extensional Query Evaluation in Probabilistic Databases". *submitted*, 2009.
4. Karp, Luby, and Madras. "Monte-Carlo Approximation Algorithms for Enumeration Problems". In *J. Algorithms*, 1989.
5. Olteanu and Huang. "Using OBDDs for Efficient Query Evaluation on Probabilistic Databases". In *SUM*, 2008.
6. Olteanu and Huang. "Secondary-Storage Confidence Computation for Conjunctive Queries with Inequalities". In *SIGMOD*, 2009.
7. Olteanu, Huang, and Koch. "SPROUT: Lazy vs. Eager Query Plans for Tuple-Independent Probabilistic Databases". In *IEEE ICDE*, 2009.
8. Olteanu, Huang, and Koch. "Approximate Confidence Computation in Probabilistic Databases". In *IEEE ICDE*, 2010.

Full Satisfiability of UML Class Diagrams (Extended Abstract)

Alessandro Artale, Diego Calvanese, and Angelica Ibáñez-García

KRDB Research Centre, Free University of Bozen-Bolzano, Italy
{artale,calvanese}@inf.unibz.it yibanez@gmail.com

Abstract. Class diagrams are one of the most important components of UML, the de-facto standard formalism for the analysis and design of software. The semantics of UML class diagrams is by now well established, and one can exploit automated reasoning tools that are based on the languages underlying their formalization to detect relevant properties, such as class satisfiability and subsumption. Among the reasoning tasks of interest, the basic one is detecting full satisfiability of a diagram, i.e., whether *all* classes and associations of the diagram can be simultaneously populated without violating any constraints of the diagram. While the complexity of class satisfiability has been studied extensively, full satisfiability received less attention. In this paper we address this problem, and establish tight upper and lower bounds for full satisfiability of UML class diagrams. Our results confirm the intuition that full satisfiability has the same computational complexity as class satisfiability.

1 Introduction

UML (Unified Modeling Language)[1] is the de-facto standard formalism for the analysis and design of software. One of the most important components of UML are *class diagrams* (UCDs), which model the information on the domain of interest in terms of objects organized in classes and associations between them (representing relations between class instances). The semantics of UCDs is by now well established, and several works propose to represent it using various kinds of formal systems, e.g., [12,11,13,4]. Hence, one can in principle reason on a UCD and formally prove properties about it. The properties that one is interested in are, e.g., subsumption between two classes, i.e., the fact that each instance of one class is necessarily also an instance of another class, satisfiability of a specific class or association in the diagram, i.e., the fact that the information encoding it in the diagram is not contradictory, and *full satisfiability* of the diagram [15], i.e., the fact that all classes and associations in the diagram are simultaneously satisfiable. The latter property is of importance since the presence of some unsatisfiable class or association actually means either that the diagram contains unnecessary information that should be removed, or that there is some modeling error that lead to the loss of satisfiability. In fact, it can be considered as the most fundamental property that should be satisfied by a UCD.

[1] http://www.omg.org/spec/UML/

The proposed formalizations of UCDs indicate that one can resort to the powerful automated inference mechanisms provided by the adopted models to verify the above mentioned properties. Such a reasoning support [9] is of importance for various tasks related to the design, maintenance, evolution, and integration of UCDs. For example, the formalization in terms of expressive Description Logics (DLs) [3] provided in [4] is well suited for this purpose. DLs are decidable logics that are specifically designed for the conceptual representation of an application domain in terms of classes and relationships between them. Representing conceptual data models by means of DLs has gathered consensus over the years, cf. [5,6,2,9,10,7,4], and allows one to exploit state-of-the-art DL reasoners [17] for inference in such models.

However, to avoid possible performance bottlenecks that could result from using a too powerful inference mechanism, and to be able to select the appropriate one to use for the above mentioned tasks, a fundamental question that needed to be addressed was that of the intrinsic complexity of reasoning on UCDs (independently of the formal tool adopted for describing them). This problem was addressed first in [4], where, somewhat surprisingly, it was shown that the simultaneous presence of multiplicity constraints and of completeness constraints on class and association hierarchies leads to ExpTime-hardness of *class satisfiability*. This result was then strengthened in [1] to UCDs[2] with simple ISA between associations (and completeness constraints on class hierarchies only).

However, no work had addressed explicitly the complexity of full satisfiability of UCDs[3]. In this paper, we fill this gap, by showing that the complexity of full satisfiability coincides with that of classical satisfiability. Our results build on the formalization of UML CDs in terms of DLs given in [4]. In fact, the upper bound is an almost direct consequence of the corresponding upper bound for UCDs derived from the DL formalization. Instead, our lower bound is more involved, as it requires a careful analysis of the corresponding proof for class satisfiability.

The rest of the paper is organized as follows. In Section 2, we briefly introduce the DL $\mathcal{ALC}$, on which we base our results, and show that full satisfiability in $\mathcal{ALC}$ is ExpTime-complete. In Section 3, we recall the formalization of UCDs. In Section 4, we provide our main results on full satisfiability of UCDs. Finally, in Section 5, we draw some conclusions.

2 Full Satisfiability in the Description Logic $\mathcal{ALC}$

We start by studying *full satisfiability* for the DL $\mathcal{ALC}$, one of the basic variants of DLs [3]. The basic elements of $\mathcal{ALC}$ are atomic concepts and roles, denoted by A and P, respectively. Complex concepts C, D are defined by the following

[2] The results in [1] are formulated in terms of the Entity-Relationship model, but they carry directly over also to UML class diagrams.

[3] An exception is [15], which provides a PSpace upper bound for full satisfiability. However, our results here show that the algorithm of [15] must be incomplete.

rules:

$$C, D \ ::= \ A \ \mid \ \neg C \ \mid \ C \sqcap D \ \mid \ \exists P.\, C$$

The semantics of $\mathcal{ALC}$, as usual in DLs, is specified in terms of an *interpretation*. An *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, with a non empty *domain* $\Delta^{\mathcal{I}}$ and an *interpretation function* $\cdot^{\mathcal{I}}$, assigns to each concept C a subset of $\Delta^{\mathcal{I}}$, and to each role name P a binary relation in $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ such that the following conditions are satisfied:

$$\begin{aligned} A^{\mathcal{I}} &\subseteq \Delta^{\mathcal{I}}, \\ (\neg C)^{\mathcal{I}} &= \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}, \\ (C \sqcap D)^{\mathcal{I}} &= C^{\mathcal{I}} \cap D^{\mathcal{I}}, \\ (\exists P.\, C)^{\mathcal{I}} &= \{a \in \Delta^{\mathcal{I}} \mid \exists b.\, (a, b) \in P^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\}. \end{aligned}$$

We use the standard abbreviations $C_1 \sqcup C_2 := \neg(\neg C_1 \sqcap \neg C_2)$, and $\forall P.\, C := \neg\exists P.\, C$, with the corresponding semantics.

An $\mathcal{ALC}$ *terminological box* (TBox) $\mathcal{T}$ is a finite set of concept inclusion axioms of the form $C \sqsubseteq D$. An interpretation $\mathcal{I}$ *satisfies* an axiom of the form $C \sqsubseteq D$ iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. A TBox $\mathcal{T}$ is *satisfiable* if there is an interpretation $\mathcal{I}$ that satisfies every axiom in $\mathcal{T}$ (such an interpretation is called a *model* of $\mathcal{T}$). A concept C is *satisfiable w.r.t. a TBox* $\mathcal{T}$ if there is a model $\mathcal{I}$ of $\mathcal{T}$ such that $C^{\mathcal{I}} \neq \emptyset$. It can be shown that TBox satisfiability and concept satisfiability w.r.t. a TBox are reducible to each other (in polynomial time). Moreover, reasoning w.r.t $\mathcal{ALC}$ TBoxes is ExpTime-complete (see e.g., [3]).

We now define the notion of *full satisfiability* of a TBox and show that for $\mathcal{ALC}$ it has the same complexity as classical satisfiability.

Definition 1 (TBox Full Satisfiability). *Let $\mathcal{T}$ be an $\mathcal{ALC}$ TBox. $\mathcal{T}$ is said to be* fully satisfiable *if there exists a model $\mathcal{I}$ of $\mathcal{T}$ such that $A^{\mathcal{I}} \neq \emptyset$, for every atomic concept A in $\mathcal{T}$.*

Lemma 2. *Concept satisfiability w.r.t. $\mathcal{ALC}$ TBoxes can be linearly reduced to full satisfiability of $\mathcal{ALC}$ TBoxes.*

Proof. Let $\mathcal{T}$ be an $\mathcal{ALC}$ TBox and C an $\mathcal{ALC}$ concept. As pointed out in [8], C is satisfiable w.r.t. $\mathcal{T}$ if and only if $C \sqcap A_{\mathcal{T}}$ is satisfiable w.r.t. the TBox $\mathcal{T}_1$ consisting of the single assertion

$$A_{\mathcal{T}} \sqsubseteq \bigsqcap_{C_1 \sqsubseteq C_2 \in \mathcal{T}} (\neg C_1 \sqcup C_2) \sqcap \bigsqcap_{1 \leq i \leq n} \forall P_i.\, A_{\mathcal{T}}$$

where $A_{\mathcal{T}}$ is a fresh new atomic concept and $P_1, \ldots, P_n$ are all the atomic roles in $\mathcal{T}$ and C. In order to reduce the problem to full satisfiability, we extend $\mathcal{T}_1$ to $\mathcal{T}_2 = \mathcal{T}_1 \cup \{A_C \sqsubseteq C \sqcap A_{\mathcal{T}}\}$, with A_C a fresh new atomic concept, and prove that

$$C \sqcap A_{\mathcal{T}} \text{ is satisfiable w.r.t. } \mathcal{T}_1 \text{ iff } \mathcal{T}_2 \text{ is fully satisfiable}$$

($\Rightarrow$) Let $\mathcal{I}$ be a model of $\mathcal{T}_1$ such that $(C \sqcap A_{\mathcal{T}})^{\mathcal{I}} \neq \emptyset$. Construct a model of $\mathcal{T}_2$, $\mathcal{J} = (\Delta^{\mathcal{I}} \cup \{d^{top}\}, \cdot^{\mathcal{J}})$, with $d^{top} \notin \Delta^{\mathcal{I}}$, such that:

$$A_{\mathcal{T}}^{\mathcal{J}} = A_{\mathcal{T}}^{\mathcal{I}}, \quad A_C^{\mathcal{J}} = (C \sqcap A_{\mathcal{T}})^{\mathcal{I}},$$
$$A^{\mathcal{J}} = A^{\mathcal{I}} \cup \{d^{top}\} \quad \text{for all atomic concepts } A \text{ in } \mathcal{T} \text{ and } C,$$
$$P^{\mathcal{J}} = P^{\mathcal{I}} \quad \text{for all atomic roles,}$$

Obviously, the extension of every atomic concept is non empty in $\mathcal{J}$. Next, we show that $\mathcal{J}$ is indeed a model of $\mathcal{T}_2$, relying on the fact (easily proved by structural induction) that $D^{\mathcal{I}} \subseteq D^{\mathcal{J}}$, for each subconcept D of concepts in $\mathcal{T}_1$. Then, it is easy to show that $\mathcal{J}$ satisfies every assertion in $\mathcal{T}_2$:

$$A_{\mathcal{T}}^{\mathcal{J}} = A_{\mathcal{T}}^{\mathcal{I}} \subseteq \Big(\prod_{C_1 \sqsubseteq C_2 \in \mathcal{T}} (\neg C_1 \sqcup C_2) \sqcap \prod_{1 \leq i \leq n} \forall P_i . A_{\mathcal{T}}\Big)^{\mathcal{I}} \subseteq$$
$$\subseteq \Big(\prod_{C_1 \sqsubseteq C_2 \in \mathcal{T}} (\neg C_1 \sqcup C_2) \sqcap \prod_{1 \leq i \leq n} \forall P_i . A_{\mathcal{T}}\Big)^{\mathcal{J}}$$
$$A_C^{\mathcal{J}} = (C \sqcap A_{\mathcal{T}})^{\mathcal{I}} \subseteq (C \sqcap A_{\mathcal{T}})^{\mathcal{J}}$$

($\Leftarrow$) Conversely, every *full model* $\mathcal{J}$ of $\mathcal{T}_2$ is also a model of $\mathcal{T}_1$ with $(C \sqcap A_{\mathcal{T}})^{\mathcal{J}} \neq \emptyset$, as $A_C^{\mathcal{J}} \subseteq (C \sqcap A_{\mathcal{T}})^{\mathcal{J}}$. □

Theorem 3. *Full satisfiability of $\mathcal{ALC}$ TBoxes is* ExpTime*-complete.*

Proof. The ExpTime membership is straightforward, as deciding full satisfiability of an $\mathcal{ALC}$ TBox $\mathcal{T}$ can be reduced to deciding satisfiability of the TBox

$$\mathcal{T} \cup \bigcup_{1 \leq i \leq n} \{\top \sqsubseteq \exists P' . A_i\},$$

where $A_1, \ldots, A_n$ are all the atomic concepts in $\mathcal{T}$, and P' is a fresh new atomic role. The ExpTime-hardness follows from Lemma 2. □

We now modify the reduction of Lemma 2 so that it applies also to *primitive* $\mathcal{ALC}^-$ TBoxes, i.e., TBoxes that contain only axioms of the form:

$$A \sqsubseteq B, \quad A \sqsubseteq \neg B, \quad A \sqsubseteq B \sqcup B', \quad A \sqsubseteq \forall P . B, \quad A \sqsubseteq \exists P . B,$$

where A, B, B' are atomic concepts, and P is an atomic role.

Theorem 4. *Full satisfiability of primitive $\mathcal{ALC}^-$ TBoxes is* ExpTime*-complete.*

Proof. The ExpTime membership follows from Theorem 3. For proving the ExpTime-hardness, we use a result in [4] showing that concept satisfiability in $\mathcal{ALC}$ can be reduced to atomic concept satisfiability w.r.t. primitive $\mathcal{ALC}^-$ TBoxes. Let $\mathcal{T}^- = \{A_j \sqsubseteq D_j \mid 1 \leq j \leq m\}$ be a primitive $\mathcal{ALC}^-$ TBox, and A_0

an atomic concept. By Lemma 2, we have that A_0 is satisfiable w.r.t. $\mathcal{T}^-$ if and only if the TBox $\mathcal{T}_2'$ containing the axioms

$$A_{\mathcal{T}^-} \sqsubseteq \prod_{A_j \sqsubseteq D_j \in \mathcal{T}^-} (\neg A_j \sqcup D_j) \sqcap \prod_{1 \leq i \leq n} \forall P_i.\, A_{\mathcal{T}^-}, \qquad A_0' \sqsubseteq A_0 \sqcap A_{\mathcal{T}^-}\,.$$

is fully satisfiable, with $A_{\mathcal{T}^-}, A_0'$ fresh new atomic concepts. $\mathcal{T}_2'$ is not a primitive $\mathcal{ALC}^-$ TBox, but it is equivalent to the TBox containing the assertions:

$$\begin{array}{lll} A_0' \sqsubseteq A_{\mathcal{T}^-} & A_{\mathcal{T}^-} \sqsubseteq \neg A_1 \sqcup D_1 & A_{\mathcal{T}^-} \sqsubseteq \forall P_1.\, A_{\mathcal{T}^-} \\ & \vdots & \vdots \\ A_0' \sqsubseteq A_0 & A_{\mathcal{T}^-} \sqsubseteq \neg A_m \sqcup D_m & A_{\mathcal{T}^-} \sqsubseteq \forall P_n.\, A_{\mathcal{T}^-}, \end{array}$$

Finally, to get a primitive $\mathcal{ALC}^-$ TBox, $\mathcal{T}_2^-$, we replace each axiom of the form $A_{\mathcal{T}^-} \sqsubseteq \neg A_j \sqcup D_j$ by $A_{\mathcal{T}^-} \sqsubseteq B_j^1 \sqcup B_j^2$, $B_j^1 \sqsubseteq \neg A_j$, and $B_j^2 \sqsubseteq D_j$, with B_j^1, B_j^2 fresh new atomic concepts, for $j = 1, \ldots, m$.

We show now that $\mathcal{T}_2'$ is fully satisfiable iff $\mathcal{T}_2^-$ is fully satisfiable:

($\Rightarrow$) Let $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ be a full model of $\mathcal{T}_2'$. We extend $\mathcal{I}$ into a full model $\mathcal{J}$ of $\mathcal{T}_2^-$. Let $\Delta^{\mathcal{J}} = \Delta^{\mathcal{I}} \cup \{d^+, d^-\}$, with $\{d^+, d^-\} \cap \Delta^{\mathcal{I}} = \emptyset$, and define $\cdot^{\mathcal{J}}$ as follows:

$$\begin{array}{rl} A_{\mathcal{T}^-}^{\mathcal{J}} = A_{\mathcal{T}^-}^{\mathcal{I}}, \quad (A_0')^{\mathcal{J}} = (A_0')^{\mathcal{I}}, & \\ A^{\mathcal{J}} = A^{\mathcal{I}} \cup \{d^+\}, & \text{for every other atomic concept in } \mathcal{T}_2', \\ (B_j^1)^{\mathcal{J}} = (\neg A)^{\mathcal{J}}, \text{ and } (B_j^2)^{\mathcal{J}} = (D_j)^{\mathcal{J}}, & \text{for each } A_{\mathcal{T}^-} \sqsubseteq B_j^1 \sqcup B_j^2 \in \mathcal{T}_2^-, \\ P^{\mathcal{J}} = P^{\mathcal{I}} \cup \{(d^+, d^+)\}, & \text{for every atomic role } P \text{ in } \mathcal{T}_2^-. \end{array}$$

It is easy to see now, that $\mathcal{J}$ fully satisfies $\mathcal{T}_2^-$.

($\Leftarrow$) Trivial. Every model of $\mathcal{T}_2^-$ is a model of $\mathcal{T}_2'$. □

3 Formalizing UML Class Diagrams

In this section, we briefly describe UCDs and provide their semantics in terms of First Order Logic (the formalization adopted here is based on previous presentations in [4,10]).

A *class* in a UCD denotes a set of objects with common features. Formally, a class C corresponds to a unary predicate C. An *association* represents a relation between instances of two or more classes. Names of associations (as names of classes) are unique in a UCD. A binary association between two classes C_1 and C_2 is graphically rendered as in Fig. 1. The *multiplicity* constraint $n_l..n_u$ written on one end of the binary association specifies that each instance of the class C_1 participates at least n_l times and at most n_u times in the association R, and the multiplicity constraint $m_l..m_u$ specifies an analogous constraint for each instance of the class C_2. When a multiplicity constraint is omitted, it is intended to be $0..*$. Formally, an association R between the classes

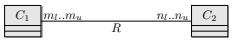


Fig. 1. Binary association

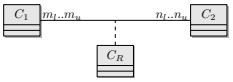


Fig. 2. Binary assoc. with related class

C_1, C_2 is captured by a binary predicate R that satisfies the FOL assertion $\forall x_1, x_2.\,(R(x_1, x_2) \rightarrow C_1(x_1) \wedge C_2(x_2))$, while multiplicities are formalized by the following FOL assertions:

$$\forall x.\,(C_1(x) \rightarrow \exists_{\geq n_l} y.\,R(x, y) \wedge \exists_{\leq n_u} y.\,R(x, y))$$
$$\forall y.\,(C_2(y) \rightarrow \exists_{\geq m_l} x.\,R(x, y) \wedge \exists_{\leq m_u} x.\,R(x, y)),$$

where we use counting quantifiers to abbreviate the FOL formula encoding the multiplicity constraints.

An association class describes properties of the association, such as attributes, operations, etc. (see Fig. 2). A binary association with a related *association class* C_R is formalized in FOL by reifying the association into a unary predicate C_R with two binary predicates P_1, P_2, one for each component of the association. We enforce the following semantics for $i = 1, 2$:

$$\begin{aligned}
\forall x.(C_R(x) &\rightarrow \exists y.\,P_i(x, y)),\\
\forall x, y.(C_R(x) \wedge P_i(x, y) &\rightarrow C_i(y)),\\
\forall x, y, y'.(C_R(x) \wedge P_i(x, y) \wedge P_i(x, y') &\rightarrow y = y'),\\
\forall y_1, y_2, x, x'.(C_R(x) \wedge C_R(x') \wedge (\textstyle\bigwedge_{i\in\{1,2\}} P_i(x, y_i) \wedge P_i(x', y_i)) &\rightarrow x = x').
\end{aligned}$$

For associations with a related class, the multiplicity constraints are formalized by the following FOL assertions:

$$\begin{aligned}
&\forall y_1.(C_1(y_1) \rightarrow \exists_{\geq n_l} x.\,(C_R(x) \wedge P_1(x, y_1)) \wedge \exists_{\leq n_u} x.\,(C_R(x) \wedge P_1(x, y_1)))\,,\\
&\forall y_2.(C_2(y_2) \rightarrow \exists_{\geq m_l} x.\,(C_R(x) \wedge P_2(x, y_2)) \wedge \exists_{\leq m_u} x.\,(C_R(x) \wedge P_2(x, y_2)))\,.
\end{aligned}$$

Generalizations (called also ISA constraints) between two classes C_1 and C specify that each instance of C_1 is also an instance of C. Several generalizations can be grouped together to form a class *hierarchy*, as shown in Fig. 3. *Disjointness* and *completeness* constraints can also be enforced on a class hierarchy, by adding suitable labels to the diagram. The class hierarchy shown in Fig. 3 is formally captured by means of the assertion $\forall x.\,C_i(x) \rightarrow C(x)$ for $i = 1, \ldots, n$. *Disjointness* among the classes $C_1, \ldots, C_n$ is expressed by $\forall x.\,C_i(x) \rightarrow \bigwedge_{j=i+1}^{n} \neg C_j(x)$ for $i = 1, \ldots, n-1$. Finally, the *completeness* constraint expressing that each instance of C is an instance of at least one of $C_1, \ldots, C_n$ is given by $\forall x.\,C(x) \rightarrow \bigvee_{i=1}^{n} C_i(x)$.

We can also have generalization between associations and between association classes with the obvious subset semantics as for generalization between classes. Finally, we do also allow for attributes associated to classes. Since the addition of attributes does not change the complexity of the satisfiability problem we do not present here attributes and their semantics.

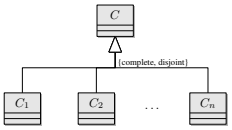


Fig. 3. A class hierarchy in UML

4 Full Satisfiability of UML Class Diagrams

Formally, three notions of UCD satisfiability have been proposed in the literature [16,4,15,14] First, *diagram satisfiability* of an UML diagram refers to the existence of a model of the diagram. Such model does not need to satisfy (populate) any class or association per se. The only condition is that all constraints are satisfied by the given model. Second, *class satisfiability* refers to the existence of a model of the diagram that satisfies (populates) a given class. Third, we can check whether there is a model of an UML diagram that satisfies *all* classes and *all* associations in a diagram. This last notion of satisfiability, referred here as *full satisfiability* and introduced in [15] is thus stronger than diagram satisfiability as a model of a diagram that satisfies all classes is, by definition, also a model of that diagram.

Definition 5 (UML Full Satisfiability). *A UCD $\mathcal{D}$ is* fully satisfiable *if there is an FOL interpretation $\mathcal{I}$ that satisfies all the constraints expressed in $\mathcal{D}$ and such that $C^{\mathcal{I}} \neq \emptyset$ for every class C in $\mathcal{D}$, and $R^{\mathcal{I}} \neq \emptyset$ for every association R in $\mathcal{D}$. We say that $\mathcal{I}$ is a* full model *of $\mathcal{D}$.*

We now address the complexity of full satisfiability for UCDs. We use the results presented in Section 2 and reduce full satisfiability of primitive $\mathcal{ALC}^-$ TBoxes to full satisfiability of UCDs. This reduction is based on the ones used in [4,1] for determining the lower complexity bound of schema satisfiability in the extended Entity-Relationship model.

Given a primitive $\mathcal{ALC}^-$ TBox $\mathcal{T}$, construct an UCD $\Sigma(\mathcal{T})$ as follows: for each atomic concept A in $\mathcal{T}$, introduce a class A in $\Sigma(\mathcal{T})$. Additionally, introduce a class O that generalizes (possibly indirectly) all the classes in $\Sigma(\mathcal{T})$ that encode an atomic concept in $\mathcal{T}$. For each atomic role P, introduce a class C_P, which reifies the binary relation P. Further, introduce two functional associations P_1, and P_2 that represent the first and second components of P. The assertions in $\mathcal{T}$ are encoded as follows:

1. For each assertion of the form $A \sqsubseteq B$, introduce a generalization between the classes A and B.
2. For each assertion of the form $A \sqsubseteq \neg B$, construct the hierarchy shown in Fig. 4.
3. For each assertion of the form $A \sqsubseteq B_1 \sqcup B_2$, introduce an *auxiliary* class B, and construct the diagram in Fig. 5.

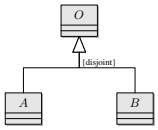


Fig. 4. Encoding of $A \sqsubseteq \neg B$

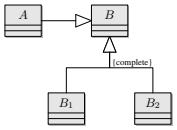


Fig. 5. Encoding of $A \sqsubseteq B_1 \sqcup B_2$

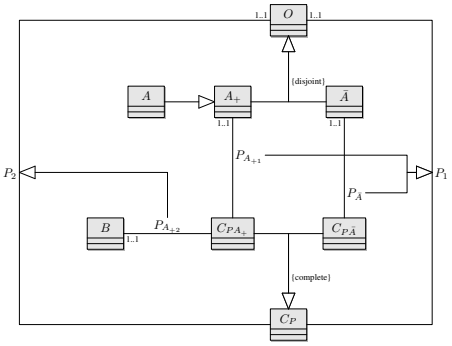


Fig. 6. Encoding of $A \sqsubseteq \forall P.\, B$

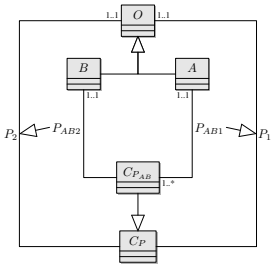


Fig. 7. Encoding of $A \sqsubseteq \exists P.\, B$

4. For each assertion of the form $A \sqsubseteq \forall P.\, B$, add the auxiliary classes C_{PA_+} and $C_{P\bar{A}}$, and the associations $P_{\bar{A}}$, $P_{A_{+1}}$ and $P_{A_{+2}}$, and construct the diagram shown in Fig. 6.
5. For each assertion of the form $A \sqsubseteq \exists P.\, B$, add the auxiliary class $C_{P_{AB}}$, and construct the diagram shown in Fig. 7.

Lemma 6. *A primitive $\mathcal{ALC}^-$ TBox $\mathcal{T}$ is fully satisfiable iff the UCD $\Sigma(\mathcal{T})$, constructed as above, is fully satisfiable.*

Proof. ($\Leftarrow$) Let $\mathcal{J} = (\Delta^{\mathcal{J}}, \cdot^{\mathcal{J}})$ be a full model of $\Sigma(\mathcal{T})$. We construct a full model $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ of $\mathcal{T}$ by taking $\Delta^{\mathcal{I}} = \Delta^{\mathcal{J}}$. Further, we define $A^{\mathcal{I}} = A^{\mathcal{J}}$ and $P^{\mathcal{I}} = (P_1^- \circ P_2)^{\mathcal{J}}$ for every concept name A and for every atomic role P in $\mathcal{T}$, respectively. Let us show that $\mathcal{I}$ satisfies every assertion in $\mathcal{T}$.

1. For assertions of the form $A \sqsubseteq B$, $A \sqsubseteq \neg B$, and $A \sqsubseteq B_1 \sqcup B_2$, the statement easily follows from the construction of $\mathcal{I}$.
2. For each assertion of the form $A \sqsubseteq \forall P.B$, let $o \in A^{\mathcal{I}} = A^{\mathcal{J}}$ and $o' \in \Delta^{\mathcal{I}} = \Delta^{\mathcal{J}}$, such that $(o, o') \in P^{\mathcal{I}}$. Since $P^{\mathcal{I}} = (P_1^- \circ P_2)^{\mathcal{J}}$, there is $o'' \in \Delta^{\mathcal{J}}$ such that $(o, o'') \in (P_1^-)^{\mathcal{J}}$, and $(o'', o') \in {P_2}^{\mathcal{J}}$. Then, $o'' \in C_P^{\mathcal{J}}$, and by the completeness constraint, $o'' \in C_{PA_+}^{\mathcal{J}} \cup C_{P\bar{A}}^{\mathcal{J}}$. We claim that $o'' \in C_{PA_+}^{\mathcal{J}}$. Suppose otherwise, then there is a unique $a \in \Delta^{\mathcal{J}}$, such that $(o'', a) \in P_{\bar{A}}^{\mathcal{J}}$ and $a \in \bar{A}^{\mathcal{J}}$. It follows from $P_{\bar{A}}^{\mathcal{J}} \subseteq P_1^{\mathcal{J}}$ and by the multiplicity constraint over C_P, that $a = o$. This rises a contradiction, because $o \in A^{\mathcal{J}} \subseteq A_+^{\mathcal{J}}$ and, $A_+^{\mathcal{J}}$ and $\bar{A}^{\mathcal{J}}$ are disjoint. Then $o'' \in C_{PA_+}^{\mathcal{J}}$. Further, there is a unique $b \in \Delta^{\mathcal{J}}$

with $(o'', b) \in P^{\mathcal{J}}_{A_{+2}}$ and $b \in B^{\mathcal{J}}$. From $P^{\mathcal{J}}_{A_{+2}} \subseteq P^{\mathcal{J}}_2$ and the multiplicity constraint on C_P, it follows that $b = o'$. Thus, we have that $o' \in B^{\mathcal{J}} = B^{\mathcal{I}}$, and therefore, $o \in (\forall P.B)^{\mathcal{I}}$.

3. For each assertion of the form $A \sqsubseteq \exists P.B$ in $\mathcal{T}$, let $o \in A^{\mathcal{I}} = A^{\mathcal{J}}$. Then, there is $o' \in \Delta^{\mathcal{J}}$ such that $(o', o) \in P^{\mathcal{J}}_{AB1}$ and $o' \in C^{\mathcal{J}}_{P_{AB}}$. Since $o' \in C^{\mathcal{J}}_{P_{AB}}$, there is $o'' \in \Delta^{\mathcal{J}}$ with $(o', o'') \in P^{\mathcal{J}}_{AB2}$ and $o'' \in B^{\mathcal{J}} = B^{\mathcal{I}}$. Then, as $P^{\mathcal{J}}_{AB2} \subseteq P^{\mathcal{J}}_2$, $P^{\mathcal{J}}_{AB1} \subseteq P^{\mathcal{J}}_1$ and $P^{\mathcal{I}} = (P^-_1 \circ P_2)^{\mathcal{J}}$, we can conclude that $(o, o'') \in P^{\mathcal{I}}$ and therefore, that $o \in (\exists P.B)^{\mathcal{I}}$.

($\Rightarrow$) Let $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ be a full model of $\mathcal{T}$, and $role(\mathcal{T})$ be the set of role names in $\mathcal{T}$. Extend $\mathcal{I}$ to a legal instantiation $\mathcal{J} = (\Delta^{\mathcal{J}}, \cdot^{\mathcal{J}})$ of $\Sigma(\mathcal{T})$, by assigning suitable extensions for the auxiliary classes and associations in $\Sigma(\mathcal{T})$. Let $\Delta^{\mathcal{J}} = \Delta^{\mathcal{I}} \cup \Gamma \cup \Lambda$, where: $\Lambda = \biguplus_{A \sqsubseteq \forall P.B \in \mathcal{T}} \{a_{A_+}, a_{\bar{A}}\}$, such that $\Delta^{\mathcal{I}} \cap \Lambda = \emptyset$, and $\Gamma = \biguplus_{P \in role(\mathcal{T})} \Delta_P$, with:

$$\Delta_P = \{(o, o') \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid (o, o') \in P^{\mathcal{I}}\} \cup \bigcup_{A \sqsubseteq \forall P.B \in \mathcal{T}} \{(a_{A_+}, b), (a_{\bar{A}}, \bar{o})\}$$

with b an arbitrary instance of B, and $\bar{o}$ an arbitrary element of $\Delta^{\mathcal{I}}$. We set $O^{\mathcal{J}} = \Delta^{\mathcal{I}} \cup \Lambda$, $A^{\mathcal{J}} = A^{\mathcal{I}}$ for each class A corresponding to an atomic concept in $\mathcal{T}$, and $C^{\mathcal{J}}_P = \Delta_P$ for each $P \in role(\mathcal{T})$. Additionally, the extensions of the associations P_1 and P_2 are defined as follows:

$$P^{\mathcal{J}}_1 = \{((o, o'), o) \mid (o, o') \in C^{\mathcal{J}}_P\}, \qquad P^{\mathcal{J}}_2 = \{((o, o'), o') \mid (o, o') \in C^{\mathcal{J}}_P\}.$$

We now show that $\mathcal{J}$ is a full model for $\Sigma(\mathcal{T})$.

1. For the portions of $\Sigma(\mathcal{T})$ due to TBox axioms of the form $A \sqsubseteq B$, $A \sqsubseteq \neg B$, and $A \sqsubseteq B_1 \sqcup B_2$, the statement follows from the construction of $\mathcal{J}$.
2. For each TBox axiom in $\mathcal{T}$ of the form $A \sqsubseteq \forall P.\, B$, let us define

$$\begin{array}{ll} A^{\mathcal{J}}_+ = A^{\mathcal{I}} \cup \{a_{A_+}\}, & \bar{A}^{\mathcal{J}} = O^{\mathcal{J}} \setminus A^{\mathcal{J}}_+, \\ C^{\mathcal{J}}_{PA_+} = \{(o, o') \in C^{\mathcal{J}}_P \mid o \in A^{\mathcal{J}}_+\}, & C^{\mathcal{J}}_{P\bar{A}} = \{(o, o') \in C^{\mathcal{J}}_P \mid o \in \bar{A}^{\mathcal{J}}\}, \\ P^{\mathcal{J}}_{A_{+1}} = \{((o, o'), o) \in P^{\mathcal{J}}_1 \mid o \in A^{\mathcal{J}}_+\}, & P^{\mathcal{J}}_{\bar{A}} = \{((o, o'), o) \in P^{\mathcal{J}}_1 \mid o \in \bar{A}^{\mathcal{J}}\}, \\ P^{\mathcal{J}}_{A_{+2}} = \{((o, o'), o') \in P^{\mathcal{J}}_2 \mid o \in A^{\mathcal{J}}_+\}. & \end{array}$$

It is not difficult to see that $\mathcal{J}$ satisfies the fragment of $\Sigma(\mathcal{T})$ as the one in Fig. 6. It remains to show that each class and each association have a non empty extension. This is clearly the case for classes that encode atomic concepts in $\mathcal{T}$. For the classes A_+, $\bar{A}$, C_{PA_+}, and $C_{P\bar{A}}$ we have that

$$a_{A_+} \in A^{\mathcal{J}}_+, \qquad a_{\bar{A}} \in \bar{A}^{\mathcal{J}}, \qquad (a_{A_+}, b) \in C^{\mathcal{J}}_{PA_+}, \qquad (a_{\bar{A}}, \bar{o}) \in C^{\mathcal{J}}_{P\bar{A}}.$$

For the associations P_1, P_2, $P_{A_{+1}}$, $P_{A_{+2}}$ and $P_{\bar{A}}$ we have that

$$((a_{A_+}, b), a_{A_+}) \in P^{\mathcal{J}}_{A_{+1}} \subseteq P^{\mathcal{J}}_1, \qquad ((a_{\bar{A}}, \bar{o}), a_{\bar{A}}) \in P^{\mathcal{J}}_{\bar{A}},$$
$$((a_{A_+}, b), b) \in P^{\mathcal{J}}_{A_{+2}} \subseteq P^{\mathcal{J}}_2.$$

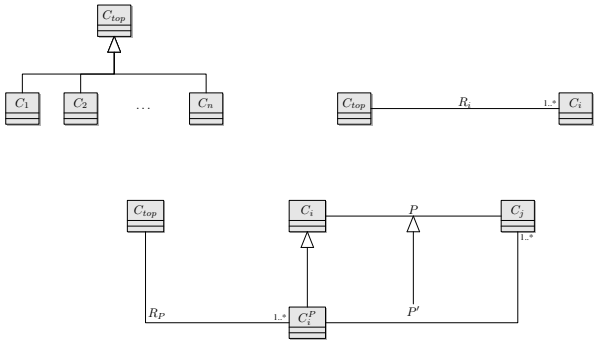


Fig. 8. Reducing UML full satisfiability to class satisfiability

3. For each TBox axiom in $\mathcal{T}$ of the form $A \sqsubseteq \exists P.B$, let us define the extensions for the *auxiliary* classes and associations as follows:

$$C^{\mathcal{J}}_{P_{AB}} = \{(o, o') \in \Delta_P \mid o \in A^{\mathcal{I}} \text{ and } o' \in B^{\mathcal{I}}\},$$
$$P^{\mathcal{J}}_{AB1} = \{((o, o'), o) \in P^{\mathcal{J}}_1 \mid (o, o') \in C^{\mathcal{J}}_{P_{AB}}\},$$
$$P^{\mathcal{J}}_{AB2} = \{((o, o'), o') \in P^{\mathcal{J}}_2 \mid (o, o') \in C^{\mathcal{J}}_{P_{AB}}\}.$$

We have that $C^{\mathcal{J}}_{P_{AB}} \neq \emptyset$ as there exists a pair $(a, b) \in \Delta_P$ with $a \in A^{\mathcal{I}}$, and $b \in B^{\mathcal{I}}$. Since $C^{\mathcal{J}}_{P_{AB}} \neq \emptyset$, we have that $P^{\mathcal{J}}_{AB1} \neq \emptyset$ and $P^{\mathcal{J}}_{AB2} \neq \emptyset$. □

Theorem 7. *Full satisfiability of UCDs is* ExpTime*-complete.*

Proof. The upper complexity bound can be established by reducing full consistency of UCDs to class consistency on UCDs, which is known to be ExpTime-complete [4]. Given a UCD $\mathcal{D}$, with classes $C_1, \ldots, C_n$, we construct the UCD $\mathcal{D}'$ by adding to $\mathcal{D}$ a new class C_{top} and a new association R_i for $i \in \{1, \ldots, n\}$. Besides, in order to ensure that every association is also populated, we consider each association P between the classes C_i and C_j such that neither C_i nor C_j is constrained to participate at least once in P. We add two new associations, R_P and P' and a new class C_i^P. Finally, we add the constraints shown in Fig. 8. Clearly, we have that $\mathcal{D}$ is fully satisfiable if and only if the class C_{top} is satisfiable.

The ExpTime-hardness follows from Lemma 6 and Theorem 4. □

5 Conclusions

This paper investigates the problem of *full satisfiability* in the context of UML class diagrams, i.e., whether *all* classes and associations of the diagram can be simultaneously populated without violating any constraints of the diagram. We show that the complexity of checking full satisfiability is ExpTime-complete,

thus matching the complexity of the classical schema satisfiability check. We show a similar result also for the problem of checking the full satisfiability of a TBox expressed in the description logic $\mathcal{ALC}$.

As a future work, we plan to extend the above results to UML class diagrams containing ad hoc subsets of the full sets of constructors. Furthermore, we intend to investigate the problem under the finite model assumption.

References

1. A. Artale, D. Calvanese, R. Kontchakov, V. Ryzhikov, and M. Zakharyaschev. Reasoning over extended ER models. In *Proc. of ER 2007*, volume 4801 of *LNCS*, pages 277–292. Springer, 2007.
2. A. Artale, F. Cesarini, and G. Soda. Describing database objects in a concept language environment. *IEEE Trans. on Knowledge and Data Engineering*, 8(2):345–351, 1996.
3. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003.
4. D. Berardi, D. Calvanese, and G. De Giacomo. Reasoning on UML class diagrams. *Artificial Intelligence*, 168(1–2):70–118, 2005.
5. S. Bergamaschi and C. Sartori. On taxonomic reasoning in conceptual design. *ACM Trans. on Database Systems*, 17(3):385–422, 1992.
6. A. Borgida. Description logics in data management. *IEEE Trans. on Knowledge and Data Engineering*, 7(5):671–682, 1995.
7. A. Borgida and R. J. Brachman. Conceptual modeling with description logics. In Baader et al. [3], chapter 10, pages 349–372.
8. M. Buchheit, F. M. Donini, and A. Schaerf. Decidable reasoning in terminological knowledge representation systems. *J. of Artificial Intelligence Research*, 1:109–138, 1993.
9. D. Calvanese, M. Lenzerini, and D. Nardi. Description logics for conceptual data modeling. In J. Chomicki and G. Saake, editors, *Logics for Databases and Information Systems*, pages 229–264. Kluwer Academic Publishers, 1998.
10. D. Calvanese, M. Lenzerini, and D. Nardi. Unifying class-based representation formalisms. *J. of Artificial Intelligence Research*, 11:199–240, 1999.
11. T. Clark and A. S. Evans. Foundations of the Unified Modeling Language. In D. Duke and A. Evans, editors, *Proc. of the 2nd Northern Formal Methods Workshop*. Springer, 1997.
12. A. Evans, R. France, K. Lano, and B. Rumpe. Meta-modelling semantics of UML. In H. Kilov, editor, *Behavioural Specifications for Businesses and Systems*, chapter 2. Kluwer Academic Publishers, 1999.
13. D. Harel and B. Rumpe. Modeling languages: Syntax, semantics and all that stuff. Technical Report MCS00-16, The Weizmann Institute of Science, Rehovot, Israel, 2000.
14. M. Jarrar and S. Heymans. Towards pattern-based reasoning for friendly ontology debugging. *Int. J. on Artificial Intelligence Tools*, 17(4):607–634, 2008.
15. K. Kaneiwa and K. Satoh. Consistency checking algorithms for restricted UML class diagrams. In *Proc. of FoIKS 2006*, pages 219–239, 2006.
16. M. Lenzerini and P. Nobili. On the satisfiability of dependency constraints in entity-relationship schemata. *Information Systems*, 15(4):453–461, 1990.

17. R. Möller and V. Haarslev. Description logic systems. In Baader et al. [3], chapter 8, pages 282–305.

A Classification Scheme for Update Propagation Methods in Deductive Databases

Andreas Behrend

University of Bonn,
Institute of Computer Science III, D-53117 Bonn
behrend@cs.uni-bonn.de

Abstract. Incremental view recomputation is a well-established research topic in deductive databases and plenty of update propagation methods have been proposed within the last two decades. All these approaches essentially apply the same propagation technique but differ in the way they are realized, the focus they provide to the induced changes and the granularity of the computed induced updates. In this paper we provide a general framework for update propagation with regards to these aspects. This allows for comparing strengths and weaknesses of different propagation methods and helps to identify potential for further refinements. As an example, we will investigate Oracle's current approach to the incremental evaluation of continuous queries and suggest possibilities for its improvement.

1 Introduction

Update Propagation (UP) has been intensively studied for many years mainly in the context of integrity checking and materialized view maintenance, e.g. [6, 8, 10, 12, 13, 15, 16, 18, 19, 21, 24, 25]. Nowadays, UP plays an important role in the context of the view-based analysis of data streams [2, 9] and forms the basis for the incremental evaluation of continuous queries (e.g. in Oracle [22]). The aim of UP is the computation of implicit changes of derived relations resulting from explicitly performed updates of the extensional fact base. As in most cases an update will affect only a small portion of the database, it is rarely reasonable to compute the induced changes by comparing the entire old and new database state. Instead, the implicit modifications should be iteratively computed by propagating the individual updates through the possibly affected views and computing their consequences.

Although all incremental methods basically employ the same propagation technique there are differences in the way they are implemented, the focus they provide to the induced updates and the granularity of the computed changes. With respect to implementation, the authors either propose propagation algorithms of their own or the application of deductive or active propagation rules. The focus of a propagation method basically describes the way how often explicit and implicit modifications are employed during the propagation process.

While a strong focus implies the application of all base and derived updates during the computation, a small focus leads to the application of a certain subset of induced changes only, allowing to reduce the complexity of the propagation process. The different granularities can be roughly classified into true, safe, and general updates [10]. True updates correspond to the real changes whereas safe updates may as well be redundant, and general updates may even be false. The different granularities result from methods for integrity checking or materialized view maintenance where the propagation process may be simplified.

The main contribution of this paper is to provide a common framework for transformation-based approaches to set-oriented update propagation. To this end, we develop deductive propagation rules in Datalog allowing for the computation of induced updates with variable focus and in an arbitrarily chosen granularity. Every transformation-based method for update propagation is reflected in the structure of our deductive propagation rules, thus allowing a comparison of the strengths and weaknesses of the different proposals. In addition, it is possible to identify possible improvements of the proposed method, e.g., by providing a better focus on the induced updates. As an example consider the current proposal for specifying a continuous query in Oracle [22]. Given a view definition for defining relation `dest`

$$\texttt{dest}(\vec{u}) \leftarrow \texttt{r}_1(\vec{x}), \texttt{r}_2(\vec{y}), \texttt{r}_3(\vec{z})$$

Oracle internally employs (the union of) the following specialized rules for incrementally computing the induced insertions $\texttt{dest}^+$ of `dest`

$$\begin{aligned}
&\texttt{dest}^+(\vec{u}) \leftarrow \texttt{r}_1^+(\vec{x}), \texttt{r}_2^{\texttt{old}}(\vec{y}), \texttt{r}_3^{\texttt{old}}(\vec{z}) \\
&\texttt{dest}^+(\vec{u}) \leftarrow \texttt{r}_1^{\texttt{new}}(\vec{x}), \texttt{r}_2^+(\vec{y}), \texttt{r}_3^{\texttt{old}}(\vec{z}) \\
&\texttt{dest}^+(\vec{u}) \leftarrow \texttt{r}_1^{\texttt{new}}(\vec{x}), \texttt{r}_2^{\texttt{new}}(\vec{y}), \texttt{r}_3^+(\vec{z})
\end{aligned}$$

where $\texttt{r}_\texttt{i}^+$ denotes the induced insertions with respect to relation $\texttt{r}_\texttt{i}$, $\texttt{r}_\texttt{i}^{\texttt{old}}$ the old state of $\texttt{r}_\texttt{i}$ before and $\texttt{r}_\texttt{i}^{\texttt{new}}$ the new state of $\texttt{r}_\texttt{i}$ after the update has been applied, respectively. Unfortunately, the authors in [22] don't give any hint how this approach is related to other well-known proposals to update propagation. Additionally, the question whether there are possible improvements of this approach remains vaguely answered. Our framework will show that this method corresponds to the computation of true updates with a small focus. Thus, a possible improvement could be to provide a stronger focus on the induced changes, e.g., by using more than one delta table $\texttt{r}_\texttt{i}^+$ within a FROM part. For an optimized evaluation of the new and old state references the Magic Sets method could be used as proposed in [1]. Additionally, other granularities than true updates could be provided by Oracle in order to indicate very fast critical constellations within the data stream which will potentially satisfy a given monitoring condition.

This paper is organized as follows: After recalling basic concepts, the theoretical framework for update propagation is provided in Section 3 by formally classifying the various kinds of update classes. Section 4 then deals with the generation of deductive propagation rules for each update class while Section 5 is concerned with the variation of the focus provided by the propagation rules. Section 6 concludes the paper.

2 Basic concepts

We assume the reader to be familiar with the notions Datalog, deductive databases and stratification. A Datalog *rule* is a function-free clause of the form $H \leftarrow L_1 \wedge \cdots \wedge L_m$ with $m \geq 1$, where H is an atom denoting the rule's head, and $L_1, \ldots, L_m$ are literals. If $A \equiv p(t_1, \ldots, t_n)$ with $n \geq 0$ is a literal, we use $\texttt{pred}(A)$ to refer to the predicate symbol p of A. If A is the head of a given rule R, we use $\texttt{pred}(R)$ to refer to the set containing the predicate symbol of A. For a set of rules $\mathcal{R}$, $\texttt{pred}(\mathcal{R})$ is defined as $\cup_{r \in \mathcal{R}} \texttt{pred}(r)$. A deductive database $\mathcal{D}$ is a tuple $\langle \mathcal{F}, \mathcal{R} \rangle$ where $\mathcal{F}$ is a finite set of facts and $\mathcal{R}$ a finite set of rules such that $\texttt{pred}(\mathcal{F}) \cap \texttt{pred}(\mathcal{R}) = \emptyset$. Within a deductive database $\mathcal{D} = \langle \mathcal{F}, \mathcal{R} \rangle$, a predicate symbol p is called derived (view predicate), if $p \in \texttt{pred}(\mathcal{R})$. The predicate p is called extensional (or base predicate), if $p \in \texttt{pred}(\mathcal{F})$.

The semantics of a deductive database $\mathcal{D}$ is defined by its well-founded model $\mathcal{M}_\mathcal{D}$. In general, the well-founded model is three-valued and includes undefined atoms. In case of a stratifiable database $\mathcal{D}$, however, the set of undefined atoms is empty and the set of negative conclusions is just given by the complement of the set of true conclusions with respect to the Herbrand base $\mathcal{H}_\mathcal{D}$ of $\mathcal{D}$. Thus, the semantics of a stratifiable database may also be represented by the set of true atoms, only. This model-based semantics is not well-suited for computing the implicit state of a given database as it defines the semantics in a non-constructive way. Therefore, various constructive methods for computing well-founded models have been proposed in the literature which can be roughly classified into bottom-up and top-down approaches. For our framework, however, the underlying rule evaluation technique is irrelevant and we tacitly assume the respective well-founded models to be computed somehow. For illustrating the introduced notations, consider the following stratifiable deductive database $\mathcal{D} = \langle \mathcal{F}, \mathcal{R} \rangle$:

$$
\begin{array}{lll}
\underline{\mathcal{R}:} & \texttt{one\_way}(\texttt{X},\texttt{Y}) \leftarrow \texttt{path}(\texttt{X},\texttt{Y}) \wedge \neg\texttt{path}(\texttt{Y},\texttt{X}) & \qquad \underline{\mathcal{F}:} \quad \texttt{edge}(1,2) \\
& \texttt{path}(\texttt{X},\texttt{Y}) \leftarrow \texttt{edge}(\texttt{X},\texttt{Y}) & \qquad\qquad\quad \texttt{edge}(2,1) \\
& \texttt{path}(\texttt{X},\texttt{Y}) \leftarrow \texttt{edge}(\texttt{X},\texttt{Z}) \wedge \texttt{path}(\texttt{Z},\texttt{Y}) & \qquad\qquad\quad \texttt{edge}(2,3)
\end{array}
$$

Relation `path` represents the transitive closure of relation `edge` while relation `one_way` selects all $\texttt{path}(\texttt{X},\texttt{Y})$-facts where Y is reachable from X but not vice versa. A stratification induces (in this case) the unique partition $\mathcal{P} = P_1 \cup P_2$ with P_1 comprising the two `path`-rules while P_2 includes the `one_way`-rule. The semantics of $\mathcal{D}$ is given by its total well-founded model $\mathcal{M}_\mathcal{D} = \mathcal{F} \cup \{\texttt{path}(1,2), \texttt{path}(2,1), \texttt{path}(2,3), \texttt{path}(1,1), \texttt{path}(2,2), \texttt{path}(1,3)\} \cup \{\texttt{one\_way}(1,3), \texttt{one\_way}(2,3)\}$. Note that in the following definitions the notation $\mathcal{D}$ will always be used to denote a stratifiable deductive database with $\mathcal{D} = \langle \mathcal{F}, \mathcal{R} \rangle$.

3 Induced Updates

In this section we introduce the syntax and semantics of modifications on extensional as well as intensional relations of a given deductive database. We refrain

from presenting a concrete update language but rather concentrate on the resulting sets of update primitives specifying insertions and deletions of individual facts. We will use the notion of *performed update* to denote the 'true' changes of base relations caused by a transaction only; that is, we restrict the set of facts to be updated to the minimal set of updates where compensation effects (given by an insertion and deletion of the same fact or the insertion of facts which already exist in the database) are already considered. Therefore, performed updates may be seen as the effect of an applied transaction.

Definition 1 (Performed Update). *A performed update u_D is a pair $\langle u_D^+, u_D^- \rangle$ where u_D^+ and u_D^- are sets of base facts with $\texttt{pred}(u_D^+ \cup u_D^-) \subseteq \texttt{pred}(\mathcal{F})$, $u_D^+ \cap u_D^- = \emptyset$, $u_D^+ \cap \mathcal{F} = \emptyset$ and $u_D^- \subseteq \mathcal{F}$. The atoms u_D^+ represent facts to be inserted into $\mathcal{D}$, whereas u_D^- contains the facts to be deleted from $\mathcal{D}$.*

We will use the notion of *induced update* to refer to the entire set of facts in which the new state of the database differs from the old state after an update of base tables has been applied. The performed updates solely refer to base relations while the induced updates additionally include their implicit consequences to the derived relations. Note that we do not consider view update requests.

Definition 2 (Induced Update). *Let $\mathcal{D}$ be a deductive database and $u_\mathcal{D}$ a performed update of $\mathcal{D}$ resulting in the new (updated) database $\mathcal{D}'$. Then $u_\mathcal{D}$ leads to an induced update $u_{D \to D'}$ from D to D' which is a pair $\langle u_{D \to D'}^+, u_{D \to D'}^- \rangle$ of sets of ground atoms such that $u_{D \to D'}^+ = \mathcal{M}_{D'} \backslash \mathcal{M}_D$ and $u_{D \to D'}^- = \mathcal{M}_D \backslash \mathcal{M}_{D'}$. The atoms $u_{D \to D'}^+$ represent the induced insertions, whereas $u_{D \to D'}^-$ consists of the induced deletions.*

The task of update propagation is to provide a description of the overall occurred modifications. Technically, such a description is given by a set of ground dynamic literals entirely including the induced update. However, it is not generally required that such a description exactly represents the sets in $u_{D \to D'}$ but may additionally contain further ground literals. In the following we will call such descriptions, characterizing an induced update more or less accurately, a *transition portrait*. Many approaches to update propagation have been developed estimating the induced update with a variable degree of accuracy. This is mainly motivated by integrity checking problems where the exact difference of the two consecutive database states is not always necessary. Instead, an overestimation of the induced changes could be used which is given by a transition portrait.

Definition 3 (Transition Portrait). *Let $u_\mathcal{D}$ be a performed update and $u_{D \to D'}$ the resulting induced update from D to D'. Any pair of sets of facts $\langle \Delta^+, \Delta^- \rangle$ overestimating $\langle u_{D \to D'}^+, u_{D \to D'}^- \rangle$, i.e., $u_{D \to D'}^+ \subseteq \Delta^+$ and $u_{D \to D'}^- \subseteq \Delta^-$, is called transition portrait.*

1. *A transition portrait is called safe iff the conditions $\Delta^+ \subseteq \mathcal{M}_{D'}$ and $\Delta^- \subseteq \mathcal{H}_{\mathcal{D} \cup \mathcal{D}'} \backslash \mathcal{M}_{D'}$ hold.*
2. *A transition portrait is called true iff the conditions $\Delta^+ = u_{D \to D'}^+$ and $\Delta^- = u_{D \to D'}^-$ hold (i.e., it just coincides with the given induced update).*

The classification of the different forms of transition portraits is helpful as it is often not necessary to compute true changes. For instance, materialized relations can be correctly maintained on the basis of safe transition portraits. Determining true induced updates is obviously more expensive than computing safe ones, as safe updates need to be verified on the new state only whereas true updates require evaluations on both, the new and old state. In contrast to this, the number of true changes is usually smaller than that of computed safe updates. Thus, there is a trade-off between the number of determined changes and the efficiency of their computation. In the following section we develop deductive propagation rules for deriving true, safe, and general transition portraits.

4 Propagation Rules

Calculating induced updates in different granularities has been already discussed in [17] and [10]. While [17] deals with positive databases only, Griefahn already proposes in [10] a framework to the propagation of insertions and deletions of base facts in stratifiable databases. In contrast to these frameworks, however, we develop new propagation rules with a different focus on individual induced updates (i.e. the number of delta literals referenced). These rules can be further refined by the so-called merging process which will be introduced later in Section 5 and allows to systematically derive new approaches to update propagation such as the one used in Oracle.

In general, update propagation methods analyze the deductive rules of a given database in order to systematically determine so-called delta relations which provide a focus on the specific changes of relations after a performed update has been applied. Thus, delta relations can be considered the practical counterpart of transition portraits. Delta relations reflect the original database schema in such a way that the performed updates are represented by extensional delta relations while derived updates (i.e. induced updates which are not performed ones) are described by rule-defined ones. For efficiency reasons we allow to reference delta relations in the body of propagation rules as well such that their evaluation is restricted to already computed induced updates. In order to abstract from negative and positive occurrences of atoms in rule bodies, we use the superscripts "$+$" and "$-$" for indicating what kind of delta relation is to be used. For a positive literal $A \equiv p(t_1, \ldots, t_n)$ we define $A^+ \equiv p^+(t_1, \ldots, t_n)$ and $A^- \equiv p^-(t_1, \ldots, t_n)$. For a negative literal $L \equiv \neg A$, we use $L^+ := A^-$ and $L^- := A^+$. For computing the derived delta relations, the explicit changes caused by a performed update have to be represented by the extensional delta relations. Thus, we generate a set of delta facts called *propagation seeds* for a performed update.

Definition 4 (Propagation Seeds). *Let $u_D = \langle u_D^+, u_D^- \rangle$ be a performed update. The set of propagation seeds* `prop_seeds`(u_D) *with respect to u_D is*

$$\texttt{prop\_seeds}(u_D) := \{\ A^\pi \mid A \in u_D^\pi \text{ and } \pi \in \{+, -\}\}.$$

Assuming a bottom-up evaluation, the propagation seeds represent the starting point from which induced updates can be computed using propagation rules.

Within these propagation rules references to both the old and new database state are necessary. We will use corresponding superscripts `old` and `new` for referring to the respective state of a given relation.

In [12] it is assumed that all views are materialized which simplifies the state evaluation process but seems to be unrealistic in practice. Therefore, the possibility has been investigated of dropping the explicit references to one of the states by deriving it from the other one and the given performed updates. The benefit of such a state simulation is that the database system is not required to store both states explicitly but may work on one state only. Rules for state simulation have been called *transition rules* in [18] and various optimizations have been proposed for their evaluation. As an example, transition rules could be incrementally defined or Magic Sets could be used for specializing them with respect to already computed induced updates as proposed in [1]. However, we refrain from discussing state simulation because of space limitations and assume the correct state determination by the database.

4.1 Propagation Rules for True Updates

A first naive construction of propagation rules for true updates can be directly derived from the definition of the true transition portrait yielding two propagation rules for each derived relation:

$$A^+ \leftarrow A^{\texttt{new}} \wedge \neg A^{\texttt{old}} \qquad\qquad A^- \leftarrow A^{\texttt{old}} \wedge \neg A^{\texttt{new}}$$

The propagation rules are called naive as they do not take the actually performed update into account. Nevertheless, they already point out the general structure inherent in any propagation rule:

1. The *derivability test* ($A^{\texttt{new}}|A^{\texttt{old}}$) is performed in order to determine whether A is derivable in the new or old state, respectively. In fact, it is responsible for calculating general updates.
2. The *effectiveness test*[1] ($\neg A^{\texttt{old}}|\neg A^{\texttt{new}}$) checks whether the fact obtained by the derivability test is not derivable in the opposite state. Hence, it checks whether the general updates obtained by the derivability test are effective.

Semantically, however, two other tasks can be identified depending on the database state a literal refers to. The *safeness test* (`new`-derivations) takes care that only updates of a safe transition portrait are derived while the *trueness test* (`old`-derivations) takes care that only updates of the true transition portrait are propagated. As already mentioned above, the disadvantage of the naive propagation rules is that the actually performed update as well as already computed induced updates are not employed in their rule bodies. For incrementally propagating true updates the derivability test ought to include references to delta relations, too. Before defining corresponding specialized propagation rules we still need to introduce a notation for so-called preserved elements $A^{\texttt{0}}$ (according

[1] The effectiveness test is called *redundancy test* in [13].

to the naming in [15]). The set $A^{\mathtt{0}}$ comprises all tuples from the old state A^{old} of A which are not deleted, i.e., $A^{\mathtt{0}} := A^{\mathtt{old}} \setminus A^{-}$.

Definition 5 (Propagation Rules for True Updates). *Let $\mathcal{R}$ be a stratifiable deductive rule set. The set of propagation rules for true updates with respect to $\mathcal{R}$ is denoted $\varphi_t(\mathcal{R})$ and is defined as follows: For each rule $A \leftarrow L_1 \wedge \ldots \wedge L_n \in \mathcal{R}$ and each body literal L_i $(i = 1, \ldots, n)$ two propagation rules of the form*

$$A^{+} \leftarrow L_i^{+} \wedge\ L_1^{\nu^1} \wedge \ldots \wedge\ L_{i-1}^{\nu^{i-1}} \wedge\ L_{i+1}^{\nu^{i+1}} \wedge \ldots \wedge\ L_n^{\nu^n} \wedge \neg A^{\mathtt{old}}$$

$$A^{-} \leftarrow L_i^{-} \wedge\ L_1^{o^1} \wedge \ldots \wedge\ L_{i-1}^{o^{i-1}} \wedge\ L_{i+1}^{o^{i+1}} \wedge \ldots \wedge\ L_n^{o^n} \wedge \neg A^{\mathtt{new}}$$

with $\nu^i \in \{+, \mathtt{0}\}$ and $o^i \in \{-, \mathtt{old}\}$ are in $\varphi_t(\mathcal{R})$. The literals $L_j^{\nu^j}$ and $L_j^{o^j}$ are called side literals of L_i

The propagation rules still perform a comparison of the `old` and `new` database state but provide the strongest focus on individual updates by applying as many delta literals L_i^{π} with $\pi \in \{+, -\}$ as possible. Specializing the derivability test in this way has been proposed in [3] but leads to the generation of $2^n - 1$ propagation rules (if n is the number of body literals). We use the strongest focus, however, in order to provide a starting point from which propagation rules of concrete proposals can be systematically derived. To this end, we propose a merging process in Section 5 which reduces the focus on delta relations leading to a smaller number of propagation rules.

The obtained propagation rules and seeds can be added to the original database yielding a safe and stratifiable database which is called *augmented database* in the following according to the naming in [18]. The safeness of propagation rules immediately follows from the safeness of the original rules. Furthermore, the propagation rules cannot jeopardize stratifiability, as delta relations are always positively referenced and hence cannot participate in any cycle involving negation. As an example, consider again the rules from Section 2 for defining the derived relation `one_way`. The corresponding propagation rules for true insertions are as follows (In the sequel, the relation symbols will be abbreviated by their first letter.):

$$\mathtt{o}^{+}(\mathtt{X},\mathtt{Y}) \leftarrow \mathtt{p}^{+}(\mathtt{X},\mathtt{Y}) \wedge \mathtt{p}^{-}(\mathtt{Y},\mathtt{X}) \wedge \neg \mathtt{o}^{\mathtt{old}}(\mathtt{X},\mathtt{Y})$$
$$\mathtt{o}^{+}(\mathtt{X},\mathtt{Y}) \leftarrow \mathtt{p}^{+}(\mathtt{X},\mathtt{Y}) \wedge \neg \mathtt{p}^{\mathtt{0}}(\mathtt{Y},\mathtt{X}) \wedge \neg \mathtt{o}^{\mathtt{old}}(\mathtt{X},\mathtt{Y})$$
$$\mathtt{o}^{+}(\mathtt{X},\mathtt{Y}) \leftarrow \mathtt{p}^{\mathtt{0}}(\mathtt{X},\mathtt{Y}) \wedge \mathtt{p}^{-}(\mathtt{Y},\mathtt{X}) \wedge \neg \mathtt{o}^{\mathtt{old}}(\mathtt{X},\mathtt{Y})$$

Note that the exponents of the delta literals for `p` are inverted, as `p` is negatively referenced in the original rule. The following proposition shows the correctness of our propagation rules and more importantly, indicates the conditions to be satisfied for the respective proof.

Proposition 1 (Correctness of Propagation Rules). *Let $\mathcal{D} = \langle \mathcal{F}, \mathcal{R} \rangle$ be a stratifiable database, $u_{\mathcal{D}}$ a performed update and $u_{D \to D'} = \langle u^{+}_{D \to D'}, u^{-}_{D \to D'} \rangle$ the corresponding induced update from $\mathcal{D}$ to $\mathcal{D}'$. Let $\mathcal{D}^p = \langle \mathcal{F} \cup \mathtt{prop\_seeds}(u_{\mathcal{D}}), \mathcal{R} \cup \varphi_t(\mathcal{R}) \rangle$ be the augmented deductive database of $\mathcal{D}$. Then the delta relations*

correctly represent the induced update $u_{D \to D'}$ *and for each relation* $p \in \mathtt{pred}(\mathcal{D})$ *the following conditions hold:*

$$p^+(\vec{t}\,) \in \mathcal{M}_{\mathcal{D}^p} \iff p(\vec{t}\,) \in u^+_{D \to D'}$$
$$p^-(\vec{t}\,) \in \mathcal{M}_{\mathcal{D}^p} \iff p(\vec{t}\,) \in u^-_{D \to D'} \ . \qquad \square$$

We omit the proof of this and the following propositions because of space limitations. However, the proposition can be easily shown by induction on the depth of proof trees with respect to D'.

4.2 Propagation Rules for Safe Updates

An update is called safe if it is reflected in the new database state, i.e., an inserted atom is present whereas a deleted atom is absent regardless whether it was included in the old state or not. Thus, propagation rules for safe updates must include a safeness test in any case but may refrain from applying a trueness test. For positive propagation this implies that the effectiveness test can be dropped. In case of negative propagation the trueness test which coincides with the derivability test cannot be completely given up as it is responsible for computing a general update. However, it may be restricted to any subset of 'side' literals which maintains the safeness of the resulting propagation rule.

Definition 6 (Propagation Rules for Safe Updates). *Let* $\mathcal{R}$ *be a stratifiable deductive rule set. The set of propagation rules for safe updates with respect to* $\mathcal{R}$ *is denoted* $\varphi_s(\mathcal{R})$ *and is defined as follows: For each rule* $A \leftarrow L_1 \wedge \ldots \wedge L_n \in \mathcal{R}$ *and each body literal* L_i $(i = 1, \ldots, n)$ *two propagation rules of the form*

$$A^+ \leftarrow L_i^+ \wedge\ L_1^{\nu^1} \wedge \ldots \wedge\ L_{i-1}^{\nu^{i-1}} \wedge\ L_{i+1}^{\nu^{i+1}} \wedge \ldots \wedge\ L_n^{\nu^n}$$

$$A^- \leftarrow L_i^- \wedge\ \mathit{Side}(L_i^-) \wedge \neg A^{\mathtt{new}}$$

are in $\varphi_s(\mathcal{R})$ *where* $\nu^i \in \{+, \mathtt{0}\}$ *and* $\mathit{Side}(L_i^-) \subseteq \{L_1^{o^1} \wedge \ldots \wedge\ L_{i-1}^{o^{i-1}} \wedge\ L_{i+1}^{o^{i+1}} \wedge \ldots \wedge\ L_n^{o^n}\}$ *is a subset of body literals of the rule with* $o^i \in \{-, \mathtt{old}\}$ *such that the effectiveness test* $\neg A^{\mathtt{new}}$ *remains safe.*

From this definition follows that the propagation rules for safe updates are range-restricted (or safe) and do not introduce any stratification problems. The propagation rules for safely updating our sample relation `one_way` would look as follows:

$$\mathtt{o}^+(\mathtt{X},\mathtt{Y}) \leftarrow \mathtt{p}^+(\mathtt{X},\mathtt{Y}) \wedge \mathtt{p}^-(\mathtt{Y},\mathtt{X}) \qquad \mathtt{o}^-(\mathtt{X},\mathtt{Y}) \leftarrow \mathtt{p}^-(\mathtt{X},\mathtt{Y}) \wedge \neg \mathtt{o}^{\mathtt{new}}(\mathtt{X},\mathtt{Y})$$
$$\mathtt{o}^+(\mathtt{X},\mathtt{Y}) \leftarrow \mathtt{p}^+(\mathtt{X},\mathtt{Y}) \wedge \neg \mathtt{p}^{\mathtt{0}}(\mathtt{Y},\mathtt{X}) \qquad \mathtt{o}^-(\mathtt{X},\mathtt{Y}) \leftarrow \mathtt{p}^+(\mathtt{Y},\mathtt{X}) \wedge \neg \mathtt{o}^{\mathtt{new}}(\mathtt{X},\mathtt{Y})$$
$$\mathtt{o}^+(\mathtt{X},\mathtt{Y}) \leftarrow \mathtt{p}^{\mathtt{0}}(\mathtt{X},\mathtt{Y}) \wedge \mathtt{p}^-(\mathtt{Y},\mathtt{X})$$

In case of positive propagation no effectiveness test is performed whereas in case of negative propagation all side literals are dropped. Note that the first negative propagation rule actually results from two negative rules with the common delta relation $\mathtt{p}^-(\mathtt{X},\mathtt{Y})$ and the dropped side literals $\mathtt{p}^+(\mathtt{Y},\mathtt{X})$ and $\neg \mathtt{p}^{\mathtt{old}}(\mathtt{Y},\mathtt{X})$, respectively.

Proposition 2 (Correctness of Safe Propagation Rules). *Let $\mathcal{D} = \langle \mathcal{F}, \mathcal{R} \rangle$ be a stratifiable database, $u_\mathcal{D}$ a performed update, $u_{D \to D'} = \langle u^+_{D \to D'}, u^-_{D \to D'} \rangle$ the corresponding induced update from $\mathcal{D}$ to $\mathcal{D}'$ and $\varphi_s(\mathcal{R})$ a set of propagation rules for safe updates. Let $\mathcal{D}^p = \langle \mathcal{F} \cup \texttt{prop\_seeds}(u_\mathcal{D}),\ \mathcal{R} \cup \varphi_s(\mathcal{R}) \rangle$ be the augmented deductive database of $\mathcal{D}$. Then the delta relations correctly represent a safe transition portrait Δ of $u_{D \to D'}$ and for each relation $p \in \texttt{pred}(\mathcal{D})$ the following conditions hold:*

$$p(\vec{t}\,) \in \Delta^+ \ \Rightarrow\ p^+(\vec{t}\,) \in \mathcal{M}_{\mathcal{D}^p} \quad \textit{and} \quad p^+(\vec{t}\,) \in \mathcal{M}_{\mathcal{D}^p} \ \Rightarrow\ p(\vec{t}\,) \in \mathcal{M}_{\mathcal{D}'}$$
$$p(\vec{t}\,) \in \Delta^- \ \Rightarrow\ p^-(\vec{t}\,) \in \mathcal{M}_{\mathcal{D}^p} \quad \textit{and} \quad p^-(\vec{t}\,) \in \mathcal{M}_{\mathcal{D}^p} \ \Rightarrow\ p(\vec{t}\,) \in \mathcal{M}_{\mathcal{D}'}$$

□

4.3 Propagation Rules for Induced Updates in the General Case

For propagating general updates, which may be neither safe or true, the truth value of the updated facts neither in the old nor in the new state is essential. Many approaches to update propagation have been introduced which completely refrain from evaluating side literals and effectiveness tests, e.g. [5, 11, 13, 14, 27]. Consequently, propagation becomes very cheap as it mostly relies on extensional or derived delta relations, only. [27] even considers the propagation of general updates with built-in predicates as the latter can be evaluated without accessing the database. However, simply omitting all side literals may lead to unsafe propagation rules. The methods mentioned above cope with this problem by providing special propagation algorithms, e.g. [5, 11, 14], by dropping the safeness requirement, e.g. [13], or by encoding unsafe variables with special ground terms such as NULL, e.g. [27].

In order to avoid the generation of non-ground facts we propose to drop all variables indicated as unbound during the evaluation. To this end, an adornment phase is used quite similar to the one employed during the Magic Sets rewriting. Within an adorned rule set each derived predicate is associated with an adornment, which is a string consisting of the symbols ?b? and ?f? representing bound and free argument positions [20] when the predicate is evaluated. The adorned rule set is derived from the original database starting from the bindings provided by extensional delta relations and using a choice of *sideways information passing (sip) strategy* [20]. A SIP strategy determines for each rule the order in which the body literals are to be evaluated and what bindings are passed on. As an example, consider the unsafe propagation rule $\texttt{p}^+(\texttt{X},\texttt{Y}) \leftarrow \texttt{e}^+(\texttt{X},\texttt{Z})$ which can be adorned as follows:

$$\texttt{p}^+_{\texttt{bf}}(\texttt{X},\texttt{Y}) \leftarrow \texttt{e}^+(\texttt{X},\texttt{Z}) \qquad\qquad \texttt{p}^+_{\texttt{ff}}(\texttt{X},\texttt{Y}) \leftarrow \texttt{e}^+(\texttt{X},\texttt{Z})$$

Relation $\texttt{p}^+$ is specialized into two relations $\texttt{p}^+_{\texttt{bf}}$ and $\texttt{p}^+_{\texttt{ff}}$ representing the two possible ways of passing bindings in this case. Given a literal L and its adornment ad, the encoded literal of L with respect to ad is denoted $\texttt{bd}_{\texttt{ad}}(\texttt{L})$ and represents the adorned version of L where all variables indicated as unbound are dropped. Encoding the adorned literals of the above example yields the following rules $\texttt{p}^+_{\texttt{bf}}(\texttt{X}) \leftarrow \texttt{e}^+(\texttt{X},\texttt{Z})$ and $\texttt{p}^+_{\texttt{ff}} \leftarrow \texttt{e}^+(\texttt{X},\texttt{Z})$ which are safe now. The derivation of a

fact, say $\mathtt{p}^+_{\mathtt{bf}}(\mathtt{c})$ indicates that all ground facts $\mathtt{p}^+_{\mathtt{bf}}(\mathtt{c},\mathtt{c}')$ of the Herbrand base $\mathcal{H}_{\mathcal{D}\cup\mathcal{D}'}$ matching with c in their first argument are considered to be an insertion. Note that the proposed encoding is not faithful (cf. [4]), since distinct non-ground facts like $\mathtt{e}^+(\mathtt{X},\mathtt{Y},\mathtt{c})$ and $\mathtt{e}^+(\mathtt{X},\mathtt{X},\mathtt{c})$ are equally represented by $\mathtt{e}^+_{\mathtt{ffb}}(\mathtt{c})$ such that the information provided by the repeated variables will get lost. However, in the context of general updates a faithful encoding is not necessary for preserving correctness.

Definition 7 (Propagation Rules for General Updates). *Let $\mathcal{R}$ be a stratifiable deductive rule set. The set of propagation rules for general updates with respect to $\mathcal{R}$ is denoted $\varphi_p(\mathcal{R})$ and is defined as follows: For each adorned rule $A \leftarrow L_1 \wedge \ldots \wedge L_n \in \mathcal{R}$, each body literal L_i $(i = 1,\ldots,n)$ and each adornment string a,a',b,b' two propagation rules of the form*

$$\mathtt{bd}_{a'}(A)^+ \leftarrow \mathtt{bd}_a(L_i)^+ \wedge \ \mathit{Side}(L_i^+)$$

$$\mathtt{bd}_{b'}(A)^- \leftarrow \mathtt{bd}_b(L_i)^- \wedge \ \mathit{Side}(L_i^-)$$

are in $\varphi_s(\mathcal{R})$ where $\nu^i \in \{+,\mathtt{0}\}$, $\mathit{Side}(L_i^+) \subseteq \{L_1^{\nu^1} \wedge \ldots \wedge L_{i-1}^{\nu^{i-1}} \wedge L_{i+1}^{\nu^{i+1}} \wedge \ldots \wedge L_n^{\nu^n}\}$ and $o^i \in \{-,\mathtt{old}\}$, $\mathit{Side}(L_i^-) \subseteq \{L_1^{o^1} \wedge \ldots \wedge L_{i-1}^{o^{i-1}} \wedge L_{i+1}^{o^{i+1}} \wedge \ldots \wedge L_n^{o^n}\}$ such that the adorned head literals $\mathtt{bd}_{\mathtt{a}'}(\mathtt{A})$ and $\mathtt{bd}_{b'}(A)$ remain safe.

Before showing the correctness of the propagation rules for general updates let us consider the rules of our running example again. In order to propagate general updates we may use the following rules:

$$\begin{array}{ll}
\mathtt{o}^+_{\mathtt{bf}}(\mathtt{X}) \leftarrow \mathtt{p}^+_{\mathtt{bf}}(\mathtt{X}) & \mathtt{o}^-_{\mathtt{fb}}(\mathtt{Y}) \leftarrow \mathtt{p}^-_{\mathtt{bf}}(\mathtt{Y}) \\
\mathtt{p}^+_{\mathtt{bf}}(\mathtt{X}) \leftarrow \mathtt{e}^+(\mathtt{X},\mathtt{Y}) & \mathtt{p}^-_{\mathtt{bf}}(\mathtt{X}) \leftarrow \mathtt{e}^-(\mathtt{X},\mathtt{Y}) \\
\mathtt{p}^+_{\mathtt{bf}}(\mathtt{X}) \leftarrow \mathtt{p}^+_{\mathtt{bf}}(\mathtt{Z}) \wedge \mathtt{e}^{\mathtt{0}}(\mathtt{X},\mathtt{Z}) & \mathtt{p}^-_{\mathtt{bf}}(\mathtt{X}) \leftarrow \mathtt{p}^-_{\mathtt{bf}}(\mathtt{Z}) \wedge \mathtt{e}^{\mathtt{old}}(\mathtt{X},\mathtt{Z})
\end{array}$$

The employed SIP strategy passes the fist argument bindings of relation e and p such that only the side literal $\mathtt{e}^{\mathtt{old}}(\mathtt{X},\mathtt{Z})$ remains within the rules. The corresp. two old state references are needed for preserving safeness of the respective rules.

Proposition 3 (Correctness of Prop. Rules for General Updates). *Let $\mathcal{D} = \langle \mathcal{F}, \mathcal{R}\rangle$ be a stratifiable database, $u_\mathcal{D}$ a performed update, $u_{D\to D'} = \langle u^+_{D\to D'}, u^-_{D\to D'}\rangle$ the corresponding induced update from $\mathcal{D}$ to $\mathcal{D}'$ and $\varphi_p(\mathcal{R})$ a set of propagation rules for general updates. Let $\mathcal{D}^p = \langle \mathcal{F} \cup \mathtt{prop\_seeds}(u_\mathcal{D}), \mathcal{R} \cup \varphi_p(\mathcal{R})\rangle$ be the augmented deductive database of $\mathcal{D}$. Then the delta relations correctly represent a general transition portrait Δ of $u_{D\to D'}$ and for each relation $p \in \mathtt{pred}(\mathcal{D})$ the following conditions hold:*

$$\begin{array}{lll}
p(\vec{t}\,) \in \Delta^+ & \Rightarrow & \textit{it exists an adornment a of p such that } \mathtt{bd}(p^+_a(\vec{t})) \in \mathcal{M}_{\mathcal{D}^p} \\
p(\vec{t}\,) \in \Delta^- & \Rightarrow & \textit{it exists an adornment a of p such that } \mathtt{bd}(p^-_a(\vec{t})) \in \mathcal{M}_{\mathcal{D}^p}
\end{array}$$

□

Method	Insertions	Deletions
Griefahn [10]	true	true
Olivé [18]	true	true
Ceri/Widom [6]	true	safe
Gupta/Mumick [12]	true	safe
Sadri et al. [23]	general	safe
Das et al. [7]	safe	general
Këchenhoff [13]	safe	general/true
Vieille et al. [26]	safe	general/safe
Griefahn et al.[11]	general	general
Lloyd et al. [14]	general	general

Fig. 1. Accuracy of transition portraits obtained in the literature

4.4 Discussion

As stated before, there is a trade-off between the granularity of induced updates, their number, and the complexity of their evaluation. Therefore, it would be useful to separately decide the granularity of induced updates for each relation. Actually, many approaches to update propagation compute transition portraits of variable granularity. However, the distinction is rarely made for individual relations but for deletions and insertions in general. For instance, the approach in [23] determines general insertions and true deletions while [7] computes safe insertions and general deletions. The derivation of safe insertions and general deletions is proposed in [13, 26] as well. However, in [26] a general deletion has to additionally pass the safeness test, if it is further propagated through a negative literal. In this case [13] even checks trueness. Our framework can be easily extended such that the updates of each individual relation is described at an arbitrary granularity. However, we refrain from showing how propagation rules of different update classes can be mixed for individual relations due to space limitations. The table in Figure 1 classifies some approaches from above according to the accuracy of computed induced updates.

5 Merging Propagation Rules

In this section the possibility of merging propagation rules is discussed for reducing the number of rules for computing induced updates. A smaller number of propagation rules implies computational advantages but has to be paid by a lower focus as the number of delta literals within the propagation rule' bodies is reduced. Let us consider the rule $\mathtt{p}(\bar{\mathtt{v}}) \leftarrow \mathtt{a}(\bar{\mathtt{x}}) \wedge \mathtt{b}(\bar{\mathtt{y}}) \wedge \mathtt{c}(\bar{\mathtt{z}})$ and its corresponding propagation rules (with omitted effectiveness tests for simplicity reasons) for propagating true insertions:

(1) $\mathtt{p}^+(\bar{\mathtt{v}}) \leftarrow \mathtt{a}^+(\bar{\mathtt{x}}) \wedge \mathtt{b}^+(\bar{\mathtt{y}}) \wedge \mathtt{c}^+(\bar{\mathtt{z}})$ (5) $\mathtt{p}^+(\bar{\mathtt{v}}) \leftarrow \mathtt{a}^{\mathtt{0}}(\bar{\mathtt{x}}) \wedge \mathtt{b}^+(\bar{\mathtt{y}}) \wedge \mathtt{c}^+(\bar{\mathtt{z}})$

(2) $\mathtt{p}^+(\bar{\mathtt{v}}) \leftarrow \mathtt{a}^+(\bar{\mathtt{x}}) \wedge \mathtt{b}^+(\bar{\mathtt{y}}) \wedge \mathtt{c}^{\mathtt{0}}(\bar{\mathtt{z}})$
(3) $\mathtt{p}^+(\bar{\mathtt{v}}) \leftarrow \mathtt{a}^+(\bar{\mathtt{x}}) \wedge \mathtt{b}^{\mathtt{0}}(\bar{\mathtt{y}}) \wedge \mathtt{c}^+(\bar{\mathtt{z}})$
(4) $\mathtt{p}^+(\bar{\mathtt{v}}) \leftarrow \mathtt{a}^+(\bar{\mathtt{x}}) \wedge \mathtt{b}^{\mathtt{0}}(\bar{\mathtt{y}}) \wedge \mathtt{c}^{\mathtt{0}}(\bar{\mathtt{z}})$
(6) $\mathtt{p}^+(\bar{\mathtt{v}}) \leftarrow \mathtt{a}^{\mathtt{0}}(\bar{\mathtt{x}}) \wedge \mathtt{b}^+(\bar{\mathtt{y}}) \wedge \mathtt{c}^{\mathtt{0}}(\bar{\mathtt{z}})$
(7) $\mathtt{p}^+(\bar{\mathtt{v}}) \leftarrow \mathtt{a}^{\mathtt{0}}(\bar{\mathtt{x}}) \wedge \mathtt{b}^{\mathtt{0}}(\bar{\mathtt{y}}) \wedge \mathtt{c}^+(\bar{\mathtt{z}})$

All combinations of delta literals and old state references are considered, leading to an exponential number of propagation rules. This number can be reduced by introducing a new state reference for a body relation p instead of its old state or its delta relation using the property $p^{new} \supseteq p^{\mathtt{0}} \cup p^+$. For instance, overestimating $c^+(\bar{z})$ in rule 1 and 2 by using $c^{new}(\bar{z})$ instead would lead to the single rule $\mathtt{p}^+(\bar{\mathtt{v}}) \leftarrow \mathtt{a}^+(\bar{\mathtt{x}}) \wedge \mathtt{b}^+(\bar{\mathtt{y}}) \wedge \mathtt{c}^{\mathtt{new}}(\bar{\mathtt{z}})$ (denoted (1,2) in the following) which subsumes the former rules 1 and 2. Since new state references are needed anyway, e.g., within the effectiveness tests of propagation rules for induced deletions, its introduction may cause only a small computational overhead. The latter results from the smaller number of delta relations within the remaining rules such that the focus on the delta facts is reduced during the computation of induced updates. On the other hand, proceeding this way ultimately leads to the generation of the single propagation rule $\mathtt{p}^+(\bar{\mathtt{v}}) \leftarrow \mathtt{a}^{\mathtt{new}}(\bar{\mathtt{x}}) \wedge \mathtt{b}^{\mathtt{new}}(\bar{\mathtt{y}}) \wedge \mathtt{c}^{\mathtt{new}}(\bar{\mathtt{z}}) \wedge \neg \mathtt{p}^{\mathtt{old}}(\bar{\mathtt{v}})$ which subsumes all other rules but provides no focus anymore.

Thus, there is a trade-off between the number of employed propagation rules and the focus they provide. Starting from a strong focus as provided by the definitions of propagation rules, various degrees of focus can be achieved by iteratively overestimating old state and delta literals. The degree of focus could be measured by the number of referenced delta literals within the remaining set of propagation rules. For instance, twelve delta literals occur within the above example of propagation rules for true insertions with a strong focus whereas the application of the merged rule (1,2), subsuming rule 1 and 2, reduces the number of delta literals by three.

The merging process itself depends on a chosen selection function which determines the next delta or old state literal to be overestimated. Note that merging is also applicable to propagation rules for safe or even general updates. Generally, a propagation rule of the form $A^+ \leftarrow L_1 \wedge \ldots \wedge\ L_i^\nu \wedge \ldots \wedge\ L_n$ with $\nu \in \{+, \mathtt{0}\}$ specifying induced insertions with an arbitrary granularity can be overestimated by a new propagation rule in which the selected literal L_i^ν is replaced by L_i^{new}. This new rule subsumes the former one which doesn't have to be considered during the propagation process anymore. As an example, consider again the specialized continuous query in Oracle SQL as presented in Section 1. The adjusted focus can be obtained by overestimating all $b^\nu(\bar{y})$ and $c^\nu(\bar{y})$ literals in the rules from 1 to 4 and by overestimating $c^\nu(\bar{y})$ in the rules 5 and 6. The resulting merged rules represent a complete set of propagation rules for true updates with a small focus as only three delta literals remain in contrast to twelve before.

Obviously, propagation rules for induced insertions can also be merged but this time the property $p^{old} \supseteq p^-$ is employed. Generally, a propagation rule of the form $A^- \leftarrow L_1 \wedge \ldots \wedge\ L_i^- \wedge \ldots \wedge\ L_n$ specifying induced insertions with an arbitrary granularity can be overestimated by a new propagation rule in which the selected literal L_i^- is replaced by L_i^{old}. This new rule subsumes the former

one again. In case of propagation rules for true deletions, proceeding this way ultimately leads to the single rule $A^- \leftarrow L_1^{old} \wedge \ldots \wedge L_i^{old} \wedge \ldots \wedge L_m^{old} \wedge \neg A^{new}$ which subsumes all other rules for induced deletions but provides no focus anymore. As already mentioned above, the specialized views as proposed for continuous queries in Oracle [22] can be obtained by merging propagation rules for true updates in a certain way. Although not stated in [22], the corresp. propagation rules of the form

$$\begin{aligned}
&\mathtt{A}^+ \leftarrow \mathtt{L}_1^+ \wedge \mathtt{L}_2^{\mathtt{old}} \wedge \mathtt{L}_3^{\mathtt{old}} \wedge \ldots \wedge \mathtt{L}_\mathtt{n}^{\mathtt{old}} \\
&\mathtt{A}^+ \leftarrow \mathtt{L}_1^{\mathtt{new}} \wedge \mathtt{L}_2^+ \wedge \mathtt{L}_3^{\mathtt{old}} \wedge \ldots \wedge \mathtt{L}_\mathtt{n}^{\mathtt{old}} \\
&\ldots \\
&\mathtt{A}^+ \leftarrow \mathtt{L}_1^{\mathtt{new}} \wedge \mathtt{L}_2^{\mathtt{new}} \wedge \mathtt{L}_3^{\mathtt{new}} \wedge \ldots \wedge \mathtt{L}_\mathtt{n}^+
\end{aligned}$$

have been already proposed in [12] for reducing the number of propagation rules. These rules provide only a small focus but represent disjoint alternatives. In fact, using a further overestimation L_i^{new} for any old state reference L_i^{old} would induce new but redundant derivations already provided by the rule which contains the literal L_i^+ in its body. A possible deficiency of Oracle's propagation rules is that for every relation the old as well as new state reference is employed. If the new state computation of a literal - say L_1^{new} is quite expensive, its evaluation can be avoided by using the 'former' L_1^+ and L_1^0 instead. On the other hand, if the new image of relation L_1 was materialized but not the old one, it would be advantageous to overestimate every occurrence of L_1^{old} by L_1^{new} and thus, get rid off all old state references of the respective relation. In [10, 15] the overestimation of every old state reference has been proposed for specifying induced insertions:

$$\begin{aligned}
&\mathtt{A}^+ \leftarrow \mathtt{L}_1^+ \wedge \mathtt{L}_2^{\mathtt{new}} \wedge \mathtt{L}_3^{\mathtt{new}} \wedge \ldots \wedge \mathtt{L}_\mathtt{n}^{\mathtt{new}} \\
&\mathtt{A}^+ \leftarrow \mathtt{L}_1^{\mathtt{new}} \wedge \mathtt{L}_2^+ \wedge \mathtt{L}_3^{\mathtt{new}} \wedge \ldots \wedge \mathtt{L}_\mathtt{n}^{\mathtt{new}} \\
&\ldots \\
&\mathtt{A}^+ \leftarrow \mathtt{L}_1^{\mathtt{new}} \wedge \mathtt{L}_2^{\mathtt{new}} \wedge \mathtt{L}_3^{\mathtt{new}} \wedge \ldots \wedge \mathtt{L}_\mathtt{n}^+
\end{aligned}$$

No old state reference except from the effectiveness test (if needed) is used but the resulting rules do not represent disjoint cases anymore. Other selection strategies for the merging process can be used to obtain an arbitrary combination of old and new state references allowing to minimize costly state simulations. This is especially important in append-only stream applications where deletions are not considered. Consequently certain state references do not occur and shouldn't be additionally introduced by propagation rules for induced insertions.

6 Conclusion

In this paper we presented a formal framework allowing to classify set-oriented approaches to update propagation which have been mainly developed in the logic programming context. This allows for comparing advantages and shortcomings of different propagation methods. In particular we discussed Oracle's current proposal to evaluating continuous queries and motivated its extension to safe or general updates. Additionally, it has been shown how the focus of the applied propagation rules can be varied by rule merging.

References

1. Behrend A., Manthey R.: *Update Propagation in Deductive Databases Using Soft Stratification.* ADBIS 2004: 22-36
2. Behrend A., Dorau C., Manthey R., and Schüller G.: *Incremental View-Based Analysis of Stock Market Data Streams.* IDEAS, 2008: 269-275.
3. Blakeley J.A., Larson P., and Tompa F.W.: *Efficiently Updating Materialized Views.* SIGMOD: 61-71, 1986.
4. Bry F.: *Query evaluation in deductive databases: Bottom-up and top-down reconciled.* Data and Knowledge Engineering (5): 289-312, 1990.
5. Bry F., Decker H., Manthey R.: *A Uniform Approach to Constraint Satisfaction and Constraint Satisfiability in Deductive Databases.* EDBT: 488-505, 1988.
6. Ceri S. and Widom J. *Deriving Incremental Production Rules for Deductive Data.* Information Systems 19(6): 467–490, 1994.
7. Das S. K. and Williams M. H. *A Path Finding Method for Constraint Checking in Deductive Databases.* DKE (4): 223–244, 1989.
8. Decker H. *Integrity Enforcement on Deductive Databases.* Expert Database Conference 1986: 381-395.
9. Ghanem T. M. et al. *Incremental Evaluation of Sliding-Window Queries over Data Streams.* IEEE TKDE, Volumne 19(1): 57–72, 2007.
10. Griefahn U. *Reactive Model Computation–A Uniform Approach to the Implementation of Deductive Databases.* PhD Thesis, University of Bonn, 1997.
11. Griefahn U. and Lüttringhaus S. *Top-down Integrity constraint Checking in Deductive Databases.* ICLP 1990: 130-143.
12. Gupta A., Mumick I. S., and Subrahmanian V. S. *Maintaining Views Incrementally.* SIGMOD 1993, pages 157–166.
13. Küchenhoff V. *On the Efficient Computation of the Difference Between Consecutive Database States.* DOOD 1991, pages 478–502.
14. Lloyd J. W., Sonnenberg E. A., and Topor R. W. *Integrity Constraint Checking in Stratified Databases.* JLP (4): 331-343 (1987).
15. Manthey R. *Reflections on Some Fundamental Issues of Rule-Based Incremental Update Propagation.* DAISD 1994: 255-276, September 19-21, Universitat Politècnica de Catalunya.
16. Martens B. and Bruynooghe M. *Integrity Constraint Checking in Deductive Databases Using a Rule/Goal Graph.* EDS 1988, pages 567–601.
17. Moerkotte G. and Karl S. *Efficient Consistency Control in Deductive Databases.* ICDT 1988, pages 118–128.
18. Olivé A. *Integrity Constraints Checking in Deductive Databases.* VLDB 1991, pages 513–523.
19. Qian X. and Wiederhold G. *Incremental Recomputation of Active Relational Expressions.* IEEE TKDE 3: 337–341 (1991).
20. Ramakrishnan, R.: *Magic Templates: A Spellbinding Approach to Logic Programs.* JLP 11(3&4): 189-216 (1991).
21. Salem K., Beyer K., Lindsay B., and Cochrane R. *How To Roll a Join: Asynchronous Incremental View Maintenance.* SIGMOD 2000: 129-140.
22. Subramanian S. et al. *Continuous Queries in Oracle.* VLDB 2007, pages 1173-1184.
23. Sadri F. and Kowalski R. A. *A Theorem Proving Approach to Database Integrity.* Foundations of Deductive Databases and Logic Programs, pages 313–362, M. Kaufmann, 1988.

24. Sköld M. and Risch T. *Using Partial Differencing for Efficient Monitoring of Deferred Complex Rule Conditions.* ICDE 1996: 392–401.
25. Urpí T. and Olivé A. *A Method for Change Computation in Deductive Databases.* VLDB 1992, August 23–27, Vancouver, pages 225–237.
26. Vieille L., Bayer P., and Küchenhoff V.: *Integrity Checking and Materialized View Handling by Update Propagation in the EKS-V1 System.* Technical Report TR-KB-35, ECRC (1991).
27. Wüthrich B.: *Detecting Inconsistencies in Deductive Databases.* Technical Report 123, ETH Zürich (1990).

Towards Automatic Schema Mapping Verification Through Reasoning*

Paolo Cappellari and Denilson Barbosa

Department of Computing Science
University of Alberta
Edmonton, AB, Canada
{cappellari,denilson}@cs.ualberta.ca

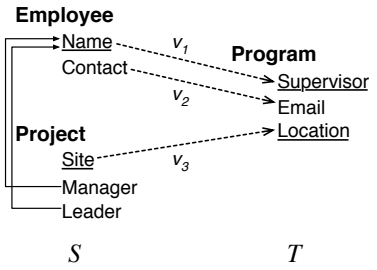
Abstract. We consider the problem of determining whether a schema mapping produces data that is *compatible* with the semantics of the target database schema. We propose a lightweight framework based annotating the schemas with conceptual meta-data expressed formally. Based on this framework, we propose a formalization of the problem, and provide algorithms that automatically verify the compatibility of mappings. Also, we discuss a strategy for automatically deriving the required semantic annotations. Finally, we discuss how our framework may lead to practical tools to help in the verification of schema mappings.

1 Introduction

Exchanging and integrating data from disparate sources are challenging and active research problems. The state-of-the-art relies on precise *schema mappings* which fully specify the relationships between elements of a *source* schema S and a *target* schema T [14], from which executable queries (or mapping programs) are derived. Tools for designing schema mappings fall in three categories: schema *matchers* [17, 18], *mapping generators* [8, 15], and *mapping verifiers* [9]. Schema matchers discover *correspondences* between pairs of individual elements of S and T. These tools are approximate in nature, often relying in the analysis of sample data. Mapping generators use the correspondences obtained by a matcher to produce the actual mappings. Finally, mapping verifiers help the designer check if the resulting schema mapping translates the data as intended. Despite the help provided by such sophisticated tools (e.g., IBM's Clio and Microsoft's BizTalk), designing mappings remains costly: some estimates suggest it takes 3 man-months to design a single mapping [19].

State-of-the-art mapping design algorithms produce mappings logically equivalent to unions of source-to-target tuple-generating-dependencies (*ST-TGDs*) [13]. Initially, they identify all so-called maximal "logical entities" in each schema; in practical terms, a logical entity is either a relation or a conjunctive query whose joins are defined over primary-key and foreign-key constraints. The *TGDs* are

* This work was supported in part by grants from the Natural Sciences and Engineering Research Council of Canada, and the Alberta Ingenuity Fund.



(a) Schemas and correspondences.

μ_1 : S:Employee(n, c), S:Project$(s, n, _)$
$\rightarrow$ T:Program(n, c, s)

μ_2 : S:Employee(n, c), S:Project$(s, _, n)$
$\rightarrow$ T:Program(n, c, s)

μ_3 : S:Employee(n, c), S:Project$(s, n, _)$
$\rightarrow$ T:Program(n, c, s)
S:Employee(n, c), S:Project$(s, _, n)$
$\rightarrow$ T:Program(n, c, s)

(b) Plausible mappings.

Fig. 1: Designing a mapping between schemas S and T. Dashed lines indicate correspondences between schema elements, while solid lines indicate foreign-key constraints; primary keys are underlined.

arrived at by pairing together logical entities from S and T that are connected through schema correspondences. If columns c_1 in S and c_2 in T are connected through a correspondence, the same variable is used (in the positions of c_1 and c_2 in the respective goals) in both the LHS and RHS of the *TGD*. On the other hand, if a column in T has no counterpart in S, a new existentially quantified variable is introduced in the RHS of the *TGD* (meaning that a new null value must be used). If there are columns corresponding to primary or foreign keys in T without corresponding columns in S, the same existentially quantified variable is used to ensure the consistency of the mapped data.

As typical large organizations have hundreds of databases, and the potentially catastrophic consequences of mapping the wrong data or incorrectly mapping the right data, the practical relevance of finding *verifiably compatible* schema mappings is paramount.

Example 1. Our running example is inspired by scenarios described in [3, 9, 16]. Figure 1(a) shows schemas S, T, some correspondences between their elements, as well as local constraints within them. For the sake of the argument, assume that the leader of a project is always an engineer and that a manager is always an administrator, and no employee is allowed to fill both roles. Further, assume instances of T stores only the technical leaders of projects. There are two maximal logical entities in S (corresponding to the two possible joins of S:Employee and S:Project) and one logical entity in T (T:Program). Figure 1(b) shows three plausible mappings that can be derived through the process outlined above. We use the standard Datalog convention of replacing variables that are not used by "don't care" symbols (_). The semantics of the mappings are as follows: μ_1 maps projects from S with their managers; μ_2 maps projects with their leaders; finally, μ_3 maps both managers and leaders. Thus, only μ_2 is compatible with the semantics of T.

Mapping Verification. The problem consists of determining whether a schema mapping $\mu : S \rightarrow T$ produces data that is *compatible with the semantics of* T. Returning to our example, the problem consists of determining which mappings in Figure 1(b) are such that only technical leaders are mapped into T:Program.

Current tools delegate the verification step to the user. Once all plausible candidate mappings are generated, the user is asked which one(s) to apply. To help in the process, these tools show examples of the target databases resulting from the application of the candidate mappings on sample data. Clearly, this does not scale: to determine the compatibility of a mapping, the designer must be an expert on both S and T or analyze many different samples of mapped data.

A step towards automatic verification is Spicy [9]. Given samples instances I^S and I^T, Spicy applies each plausible mapping to I^S and compares (statistically speaking) the resulting instances to I^T. The candidate mapping whose output is closest to I^T is chosen. Observe, however, that besides the difficulties associated with sampling, this approach relies on the problematic assumption that the semantics of a schema can be inferred from statistical correlations between data values. In our example, the compatibility of a mapping will be determined based on the correlation between names and email addresses (which are the data from S:Person that are mapped) with a person's career path (managerial vs. technical). While this assumption might hold in some cases (e.g., projects with high budget vs. projects with low budget), it clearly is not general.

Contributions. We propose a generic and lightweight framework in which to formulate and solve the mapping verification problem (Section 2). In our vision, both S and T are loosely annotated with semantic information prior to, and independently of, any mappings are designed. We verify the compatibility of a schema mapping by translating each of its *ST-TGDs* into a statement relating the conceptual description associated with relations in S and T. The verification problem is reduced to the satisfiability of the resulting statements. We rely on Description Logics [6] and the associated reasoners in our approach. Also, we explore the possibility of automatically deriving the semantic annotations necessary for our framework (Section 3).

2 The Mapping Verification Problem

Recall that our goal is determining, given two schemas S and T, whether a mapping $\mu : S \rightarrow T$ produces data that is *compatible with the semantics of* T. As stated, the definition of the problem is appealing, but also too broad. We now make it more precise.

It is worth distinguishing, however, the different aspects of the problem, some of which are partially addressed by mapping design tools. Semantic heterogeneity is a long-standing obstacle to automating data exchange and integration which is inevitable due to the subjective way in which humans abstract the real world when designing computer applications and the databases they use [1]. A correct schema mapping must resolve:

$$
\begin{aligned}
\mathit{Person} &\sqsubseteq \exists \mathit{hasName}.\mathsf{String} \sqcap \exists \mathit{hasEmail}.\mathsf{String} \\
\mathit{Woman} &\equiv \mathit{Person} \sqcap \neg \mathit{Man} \\
\mathit{Man} &\equiv \mathit{Person} \sqcap \neg \mathit{Woman} \\
\mathit{Employee} &\sqsubseteq \mathit{Person} \\
\mathit{Engineer} &\sqsubseteq \mathit{Employee} \sqcap \neg \mathit{Administrator} \\
\mathit{Administrator} &\sqsubseteq \mathit{Employee} \sqcap \neg \mathit{Engineer} \\
\mathit{Manager} &\equiv \mathit{Administrator} \sqcap \exists \mathit{manages}.\mathit{Project} \\
\mathit{Leader} &\equiv \mathit{Engineer} \sqcap \exists \mathit{leads}.\mathit{Project} \\
\mathit{Project} &\sqsubseteq \exists \mathit{hasLocation}.\mathsf{String} \\
\mathit{SeniorLeader} &\sqsubseteq \mathit{Leader} \sqcap \geq 3\ \mathit{leads}
\end{aligned}
$$

Fig. 2: Simple conceptual model describing persons and career paths.

(1) discrepancies in data representations across schemas (e.g., salaries expressed in Euros versus Canadian Dollars);
(2) discrepancies in schematic representations of the domain (e.g., the project's manager's email stored in the project table versus the employee table);
(3) *semantic discrepancies* across schemas (e.g., whether a manager can fill the role of a technical leader of a project).

While all three aspects above are part of the mapping verification problem, it is worth noting that current schema matchers and mapping design tools substantially address most of syntactic aspects (point 1) and schematic discrepancies (point 2). The semantic discrepancies, on the other hand, are not properly addressed in the state-of-the-art and are the main focus of our work.

We follow the Description Logic syntax and terminology of [6].

2.1 Semantic Annotations

For the sake of completeness, we revisit basic necessary notions of relational databases [2] and ontologies in Description Logics [6]. A *relational schema* is a set of relation schemes; each relation (scheme) $r(\boldsymbol{a})$, where $\boldsymbol{a} = a_1, \ldots, a_n$ is a list of attributes, and each a_i is associated with *domain* d_i, $i \leq i \leq n$. We say a_i is the attribute of r at position i. An ontology $\mathcal{O} = \{\mathcal{T}, \mathcal{R}\}$ is a finite set $\mathcal{T}$ of terms denoting *concepts* and a finite set $\mathcal{R}$ of binary *relationships* among terms.

Definition 1. *Let $r(\boldsymbol{a})$, $\boldsymbol{a} = a_1, \ldots, a_n$ be a relation, and $\mathcal{O} = \{\mathcal{T}, \mathcal{R}\}$ and ontology. A* semantic annotation mapping *for r is a partial function*

$$\alpha_r : \{1, \ldots, n\} \to \mathcal{T}.$$

$$\alpha_{\mathsf{S:Employee}} : \{1 \rightarrow \mathit{Employee},\ 2 \rightarrow \mathit{Employee}\}$$

$$\alpha_{\mathsf{S:Project}} : \{1 \rightarrow \mathit{Project},\ 2 \rightarrow \mathit{Manager},\ 3 \rightarrow \mathit{Leader}\}$$

$$\alpha_{\mathsf{T:Program}} : \{1 \rightarrow \mathit{Leader},\ 2 \rightarrow \mathit{Leader},\ 3 \rightarrow \mathit{Project}\}$$

Fig. 3: Semantic annotations for the schemas in Figure 1 using the conceptual model in Figure 2 as reference.

By defining α_r to be partial, we do not require that every attribute be annotated in our framework. Also, note we define α_r to assign a *concept* to each attribute in r, ignoring properties (especially data properties which naturally correspond to attributes of relations). This simplification is possible because, as mentioned above, state-of-the-art schema matchers and mapping design tools already take into account data type compatibility; thus, we choose to assume there is no need to reason at the property level.

Definition 2. *Let $S = \{r_1, \ldots, r_k\}$ be a relational schema with k relations, such that each r_i is accompanied by a semantic annotation mapping α_i. We call a set $\alpha^S = \{\alpha_{r_1}, \ldots, \alpha_{r_k}\}$ a semantic annotation for S.*

For simplicity, we assume $\mathcal{O}$ contains a universal concept, denoted $\top$ which subsumes every other concept in $\mathcal{O}$ (this corresponds to the concept often referred to as "Thing" in the knowledge representation terminology).

Example 2. Returning to our running example, consider the simple ontology in Figure 2 describing concepts related to persons (which can be either *Man* or *Woman*), and their career paths: employees (each of which is also a *Person*) can follow either the administrative career path (possibly becoming a *Manager* of a project) or the engineering career path (possibly becoming a *Leader* of a project). For simplicity, we model a *Person* as an entity that has a name and an email address (these being the *data properties* of an entity *Person*), and a *Project* as an entity that has a location.

The semantic annotations in Figure 3 describe the intended semantics as described in our motivating example (recall Example 1). In particular, note that while instances of S may store both managers and engineers, T allows for engineers only.

2.2 Representing Schema Mappings

Recall that a *tuple-generating-dependency* (*TGD*) over a relational schema S is an expression of the form $(\forall \boldsymbol{x})(q_S(\boldsymbol{x}) \rightarrow (\exists \boldsymbol{y}) q_T(\boldsymbol{x}, \boldsymbol{y}))$, where $q_S(\boldsymbol{x})$ is a conjunction of atoms over the set of variables $\boldsymbol{x}$, every variable in $\boldsymbol{x}$ appears in $q_S(\boldsymbol{x})$, and $q_T(\boldsymbol{x}, \boldsymbol{y})$ is a conjunction of atoms over $\boldsymbol{x}, \boldsymbol{y}$. An atom is an expression of the form $r(\boldsymbol{t})$, where r is a relation symbol from S and $\boldsymbol{t}$ is a tuple of variables.

As mentioned earlier, state-of-the-art schema mappings tools produce mappings logically equivalent to *source-to-target TGDs* (*ST-TGDs*) as defined in [13, 14], which involve schemas S and T and in which $q_S(\boldsymbol{x})$ is a conjunctive query (CQ) using relations from S only and $q_T(\boldsymbol{x}, \boldsymbol{y})$ is a CQ T. For simplicity, we will use the conventional Datalog notation to express conjunctive queries henceforth, and leave the quantifiers implicit.

Examples of schema mappings are given in Figure 1(b).

2.3 Assigning Semantics to Variables

Let S be a relational schema and α^S its semantic annotation into an ontology $\mathcal{O}$. We interpret S as a representation of (the properties of) objects modelled by α^S and $\mathcal{O}$. For instance, Figure 3 describes S:Employee as a relation storing two properties of objects of class *Employee* (as defined in $\mathcal{O}$). Similarly, S:Project is described as a relation in which each triplet it contains has properties of a *Project*, a *Manager* and a *Leader*, in this order. For a mapping μ to be compatible with the semantics of T, it must be the case that if an *ST-TGD* $\tau \in \mu$ maps the i-th attribute of relation $r_s \in S$ into the j-th attribute of $r_t \in T$ then the associated concepts must also be compatible.

Note that for a very simple *TGD*, in which the LHS and RHS consist of a single sub-goal (i.e., relation), we must simply check that $\alpha_{r_s}(i)$ is compatible with $\alpha_{r_t}(j)$. However, for mappings involving conjunctive queries the situation is different. Consider mapping μ_1 in Figure 1(b), and note that it is incompatible with the semantics of T because it attempts to map managers into T:Program, where only technical leaders are expected. The fact that managers are mapped follows from the join between S:Employee and S:Project on variable n (the employee's name). The incompatibility follows because the *variable* n is associated with a manager in the LHS of the *TGD* while it is associated with a technical leader in the RHS of the *TGD*. In the jargon of mapping design: in the "logical entity" defined in the LHS of the rule, n is associated with a concept that is incompatible with its counterpart in the "logical entity" defined in the RHS.

In the remainder of the section we show how to obtain the concept in $\mathcal{O}$ associated with variable v in CQ q, which we will denote by $\Gamma_q(v)$.

Using Functional Dependencies. In the discussion about μ_1 above, it is evident that n (the employee's name) in the LHS of the mapping refers to the name of a manager. This follows from using n in the join. Note that this implies that variable c in the mapping rule refers to the contact information of a manager, instead of an arbitrary employee (as indicated by the semantic annotations). This is the case because of the functional dependency Name $\rightarrow$ Contact defined in S:Employee (recall Name is the primary key of the relation).

For convenience, we will represent every functional dependencies in terms of the positions of the attributes involved in them. Without loss of generality, we only consider functional dependencies of the from $a_1, \ldots, a_n \rightarrow b$, where $b \notin \{a_1, \ldots, a_n\}$.

Definition 3. *Let $r(\boldsymbol{a})$, $\boldsymbol{a} = a_1, \ldots, a_n$, be a relation and $a_{i_1}, \ldots, a_{i_k} \rightarrow a_j$ be a functional dependency over $\boldsymbol{a}$. The* dependency set *of position j in r, denoted $\delta_r(j)$ is the set $\{i_1, \ldots, i_k\}$.*

We consider also the dependencies among variables appearing in the same predicate of a CQ:

Definition 4. *Let q be a conjunctive query, and $r(\boldsymbol{v})$, $\boldsymbol{v} = v_1, \ldots, v_n$, a predicate in q. Let v be a variable in q that appears in position i in $r(\boldsymbol{v})$. The dependency set of v in r, $\Delta_r(v)$ is the set of variables $u_1, \ldots, u_k$ such that*

- *u_j is a variable in q that appears at position j in $r(\boldsymbol{v})$;*
- *$j \in \delta_r(i)$*

In the example of μ_1 in Figure 1(a), we have that $\Delta_{\mathsf{S:Employee}}(c) = \{n\}$.

The next two definitions associate concepts with variables in a predicate of a CQ as well as the entire CQ, allowing for multiple occurrences of the same variable. First, we define the concept of a variable within a single predicate of the CQ, considering only the semantic annotations for the schema:

Definition 5. *Let $r(\boldsymbol{a})$, $\boldsymbol{a} = a_1, \ldots, a_n$, be a predicate in a CQ, in which variable v appears; the concept of v in r* at position *j is defined as*

$$\gamma_r(v, j) = \begin{cases} \alpha_r(j) & \text{if } \alpha_r(j) \text{ is defined,} \\ \top & \text{otherwise} \end{cases}$$

Now let $j_1, \ldots, j_k$ be all positions of r where v appears; the concept of v in r *is:*

$$\gamma_r(v) = \begin{cases} \gamma_r(v, j_1) \sqcap \ldots \sqcap \gamma_r(v, j_k) & \text{if } v \text{ appears in } r \\ \top & \text{otherwise} \end{cases}$$

We extend to CQs, taking into account dependencies among variables:

Definition 6. *Let $q = r_1(\boldsymbol{a_1}), \ldots, r_k(\boldsymbol{a_k})$ be a CQ in which variable v appears. The* concept of v in q, *$\Gamma_q(v)$ is defined as*

$$\Gamma_q(v) = \gamma'_{r_1}(v) \sqcap \ldots \sqcap \gamma'_{r_k}(v),$$

where $\gamma'_{r_i}(v) = \gamma_{r_i}(v) \sqcap \prod_{u \in \Delta_{r_i}(v)} \Gamma_q(u)$

Returning to our running example and μ_1 in Figure 1(b), Definition 6 gives us the following:

$$\begin{aligned} \Gamma_{q_S}(n) &= \mathit{Employee} \sqcap \mathit{Manager} & \Gamma_{q_T}(n) &= \mathit{Leader} \\ \Gamma_{q_S}(c) &= \mathit{Employee} \sqcap \mathit{Manager} & \Gamma_{q_T}(c) &= \mathit{Leader} \\ \Gamma_{q_S}(s) &= \top \sqcap \mathit{Project} & \Gamma_{q_T}(s) &= \mathit{Project} \end{aligned}$$

Note that Definition 6 is recursive, and, thus, makes sense only when there are no cyclic dependencies among variables.

Definition 7. *Let $q = r_1(\boldsymbol{v_1}), \ldots, r_k(\boldsymbol{v_k})$ be a CQ in which distinct variables v, u appear; q is* cycle-free *if there are no subgoals r_i, r_j in q such that $u \in \Delta_{r_i}(v)$ and $v \in \Delta_{r_j}(u)$.*

Input: variable v, CQ q over S, annotation $\alpha^S = \{\alpha_{r_1}, \ldots, \alpha_{r_n}\}$
Output: $\Gamma_q(v)$

```
1  Let γ = ⊤;
2  foreach subgoal r_i(v_i) in q do
3  |  if Δ_{r_i}(v) ≠ ∅ then
4  |  |  foreach u ∈ Δ_{r_i}(v) do
5  |  |  |  γ = γ ⊓ Γ_q(u)
6  |  |  end
7  |  end
8  |  Let j_1, ..., j_k be the positions where v appears in r_i(a_i);
9  |  foreach j in j_1, ..., j_k do
10 |  |  γ = γ ⊓ α_{r_i}(j)
11 |  end
12 end
13 return γ
```

Algorithm 1: Algorithm for computing $\Gamma_q(v)$.

2.4 Semantic Compatibility of Mappings

We can now formally state the problem of semantic verification of mappings: let S, T be relational schemas, α^S, α^T be their respective semantic annotations w.r.t. $\mathcal{O}$, and $\mu = \{\tau_1, \ldots, \tau_n\}$ be a schema mapping. We say that:

Definition 8. *ST-TGD $\tau_i \in \mu = (q_S(\boldsymbol{x}) \rightarrow q_T(\boldsymbol{x}, \boldsymbol{y}))$ is* compatible *with the semantics of T iff, for every $v \in \boldsymbol{x}$ in q we have that*

$$\mathcal{O} \models \Gamma_{q_S}(v) \sqsubseteq \Gamma_{q_T}(v).$$

In other words, an *ST-TGD* in a mapping is compatible with the semantics of T if it maps concepts (by sharing variables in it LHS and RHS) in a way that does not contradict the shared conceptual model $\mathcal{O}$.

The *semantic verification problem* of a schema mapping μ between schemas S and T, given semantic annotations to the same conceptual model consists of checking whether each *ST-TGD* in μ is compatible with the semantics of T:

Definition 9. *A schema mapping $\mu : S \rightarrow T = \{\tau_1, \ldots, \tau_n\}$ is compatible with the semantics of T is every $\tau_i \in \mu$ is compatible with the semantics of T.*

2.5 Algorithms

Algorithm 1 computes $\Gamma_q(v)$, for a given CQ q over S annotated with α^S.

Proposition 1. *If q is cycle-free, Algorithm 1 always terminates.*

In the sequel, we consider mappings with *ST-TGDs* containing cycle-free CQs only.

Assuming that α and $\Delta_{r_i}(v)$ are given (note that $\Delta_{r_i}(v)$ can be computed in polynomial time on the number of attributes in r_i and the number of functional dependencies defined over r_i) and also that they can be accessed in constant time, we have the following:

Proposition 2. *Algorithm 1 computes $\Gamma_q(v)$ in time $O(l^2\,|q|)$, where $|q|$ is the number of predicates in q, l is the maximum number of variables in any such predicate.*

A straightforward algorithm for checking semantic compatibility of a mapping $\mu : S \rightarrow T$ given annotations α^S and α^T into a common conceptual model $\mathcal{O}$ is as follows. For every *ST-TGD* $\tau_i \in \mu$ of the form $\tau_i = (q_S(\boldsymbol{x}) \rightarrow q_T(\boldsymbol{x}, \boldsymbol{y}))$, and for every variable $v \in \boldsymbol{x}$ produce a verification statement $\sigma_{\tau_i,v} = \Gamma_{q_S}(v) \sqsubseteq \Gamma_{q_T}(v)$ and check whether $\mathcal{O} \models \sigma_{\tau_i,v}$.

Example 3. Returning to our first example of Figure 1 using the annotations in Figure 3, we have that to verify mapping μ_1 the following verification statements are produced:

$$\text{Variable } n : \mathit{Employee} \sqcap \mathit{Manager} \sqsubseteq \mathit{Leader} \tag{1}$$
$$\text{Variable } c : \mathit{Employee} \sqcap \mathit{Manager} \sqsubseteq \mathit{Leader} \tag{2}$$
$$\text{Variable } s : \mathit{Project} \sqcap \top \sqsubseteq \mathit{Project} \tag{3}$$

Similarly, the following statements need to be tested when verifying μ_2:

$$\text{Variable } n : \mathit{Employee} \sqcap \mathit{Leader} \sqsubseteq \mathit{Leader} \tag{4}$$
$$\text{Variable } c : \mathit{Employee} \sqcap \mathit{Leader} \sqsubseteq \mathit{Leader} \tag{5}$$
$$\text{Variable } s : \mathit{Project} \sqcap \top \sqsubseteq \mathit{Project} \tag{6}$$

Thus, because μ_3 contains μ_1 (and some verification statements of mu_1 do not entail from $\mathcal{O}$), only μ_2 is deemed compatible with the semantics of T.

Next, we discuss some aspects of the computational complexity of checking the compatibility of a schema mapping.

Proposition 3. *Checking the compatibility of mapping $\mu : S \rightarrow T$ requires verifying $O(m\,|\mu|)$ statements, where $|\mu|$ is the number of ST-TGDs in μ and m is the maximum number of variables in any such ST-TGD.*

The following is a Corollary of Proposition 2:

Corollary 1. *When checking the compatibility of mapping $\mu : S \rightarrow T$, the maximum number of terms in a verification statement is polynomial in the maximum number of predicates among ST-TGDs in μ, and the highest number of variables among them.*

While both the number of verification statements and their length are bounded by polynomials in the size of the input, the full computational of checking the compatibility of schema mappings depends on the cost of checking the entailment of the verification statements against $\mathcal{O}$. Of course, this depends on several

factors: (a) the number of statements generated (b) the complexity of such statements (in terms of constructs that are used); and (c) the expressiveness of the language used to express $\mathcal{O}$. Indeed, the problem is decidable only for fragments of first-order logic for which entailment is decidable.

We advocate the use of expressive, yet practical families of Description Logics, such as $\mathcal{SHOIN}(\mathcal{D})$ (which subsumes the standard OWL-DL[1]), or $\mathcal{SHOIQ}$ (the conceptual model used in our running example—Figure 2—is an example of an ontology in this dialect). While checking entailment in such DLs is hard (concept satisfiability is know to be NEXPTIME-complete for them [21, 22]), substantial efforts have been put into developing efficient reasoning algorithms that work well in practice. Two known fast reasoners that have been used as the basis for practical implementations on large ontologies are Fact++ [23] and Pellet [20]. The latter, in fact, is aimed at Semantic Web [7] applications and to deal with ontologies in the OWL-DL standard.

3 Obtaining Semantic Annotations

The framework discussed in the previous section provides an automatic way of checking the semantic compatibility of schema mappings given semantic annotations for the schemas involved. The approach has even more practical utility if one can minimize or fully eliminate the human involvement in obtaining such annotations. We exploit this possibility next.

One prominent alternative in automatically deriving annotations is to rely on schema *alignments*, as proposed by, for instance, [4]. Such alignments are mappings between the relational schema and the conceptual model, and have been shown to help in obtaining more descriptive "logical entities" (recall the discussion in Section 1), yielding *better* schema mappings (in the sense that users in a case study preferred the mappings produced by their tool compared to those produced by Clio). Most importantly, considerable progress has been made in providing effective semi-automatic tools for discovering such alignments and expressing them formally [5], which is encouraging for the development of practical tools based on our framework.

In [4], the alignments are given as "local-as-view" (LAV) mappings of the form $r(\boldsymbol{x}) \rightarrow \exists \boldsymbol{y} \phi(\boldsymbol{x}, \boldsymbol{y})$ where r is a relation over $\boldsymbol{x}$; $\boldsymbol{y}$ is a set of variables and and ϕ is a conjunctive formula over $\mathcal{O}$. Each mapping annotates a single table in the database schema with conceptual information from the domain ontology, to guide the schema mapping algorithm in search of a better mapping.

Example 4. To express the semantics of the target schema T, which was intended for storing data about projects and their technical leaders, using the ontology in Figure 2 and the notation of [4] one could use the statements in Figure 4(a). Similarly, one could express the semantics of the source schema S, leading to the two logical entities described as in Figure 4(b).

[1] http://www.w3.org/TR/owl-guide/.

$$\mathsf{T{:}Program}(x, y, z) \rightarrow \exists l, p\, \mathcal{O}{:}\, Leader(l), \mathcal{O}{:}\, Project(p), \mathcal{O}{:}\, leads(l, p), \mathcal{O}{:}\, hasName(l, x), \mathcal{O}{:}\, hasEmail(l, y), \mathcal{O}{:}\, hasLocation(p, z)$$

(a) Alignment of T.

$$\mathsf{S{:}Employee}(x, y) \rightarrow \exists e\, \mathcal{O}{:}\, Employee(e), \mathcal{O}{:}\, hasName(e, x), \mathcal{O}{:}\, hasEmail(e, y)$$
$$\mathsf{S{:}Project}(x, y, z) \rightarrow \exists p, l, m\, \mathcal{O}{:}\, Project(p), \mathcal{O}{:}\, Leader(l), \mathcal{O}{:}\, Manager(m), \mathcal{O}{:}\, hasLocation(p, x), \mathcal{O}{:}\, hasName(m, y), \mathcal{O}{:}\, hasName(l, z)$$

(b) Alignment of S.

Fig. 4: LAV alignments of schemas in Figure 1(a) into $\mathcal{O}$.

Input: alignment $r(\boldsymbol{x}) \rightarrow \exists \boldsymbol{y} \phi(\boldsymbol{x}, \boldsymbol{y})$
Output: α_r

```
foreach i : 1...n do
    Let v_i ∈ x be the variable at position i in r(x);
    Let γ = ⊤;
    foreach predicate O:C(v_i) in φ do
        γ = γ ⊓ O:C
    end
    foreach predicate O:p(u, v_i) in φ do
        foreach predicate O:C(u) do
            γ = γ ⊓ O:C
        end
    end
    α_r(i) = γ;
end
return α_r
```

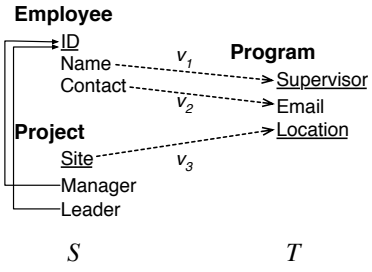
Algorithm 2: Algorithm for computing α_r from a LAV alignment.

In this work, we assume that there is at most one annotation alignment for each relation in S or T.

Deriving Annotations from LAV Alignments. Deriving semantic annotations from LAV alignments consists of parsing the mapping statements as in Algorithm 2. For each column in the relation schema, we identify the variable in the LHS of the LAV alignment rule and parse the RHS to find the concept associated with it. For instance, Applying Algorithm 2 on the LAV alignments in Figure 4 produces the annotations in Figure 3.

Proposition 4. *Algorithm 2 computes α_r in $O(|r|\,|\phi|)$ time, where $|r|$ is the number of attributes in r and $|\phi|$ is the number terms in ϕ.*

Uninterpreted Attributes. So far we have considered schemas S, T in which every attribute corresponds to a *property* of an object (e.g., the name of an employee). We call such attributes *interpreted* as their meaning is defined by $\mathcal{O}$. When all attributes are interpreted, we have that in every alignment statement $r(\boldsymbol{x}) \rightarrow$



μ_4 : S:Employee(i, n, c), S:Project$(s, _, i)$
$\rightarrow$ T:Program(n, c, s)

(a) Schema mapping.

S:Employee$(x, y, z) \rightarrow \mathcal{O}$: *Employee*$(x)$,
$\mathcal{O}$: *hasName*(x, y),
$\mathcal{O}$: *hasEmail*(x, z)
S:Project$(x, y, z) \rightarrow \exists p\, \mathcal{O}$: *Project*$(p)$,
$\mathcal{O}$: *Manager*(y)
$\mathcal{O}$: *SeniorLeader*(z),
$\mathcal{O}$: *Woman*(z),
$\mathcal{O}$: *hasLocation*(p, x)

(b) Alignment.

Fig. 5: Schema mapping and alignment using uninterpreted attributes.

$\exists \boldsymbol{y} \phi(\boldsymbol{x}, \boldsymbol{y})$, every variable v used in a predicate of the form $\mathcal{O}$: $C(v)$ in ϕ, where C is a concept, appears in $\boldsymbol{y}$.

Often, database administrators introduce artificial primary keys that effectively serve as *tuple identifiers*, for performance reasons. We call such attributes *uninterpreted* as they have not intrinsic meaning. Without loss of generality, assume that the values of the uninterpreted attributes are defined in a way that uniquely identifies the entities they belong to within the database instance (i.e., an uninterpreted attribute that identifies an employee is never used to identify a project). Now the variable in the LHS of the LAV alignment associated with the uninterpreted attribute can be used as object identifier in the RHS of the mapping, without the need for existentially quantified variables as before. Figure 5, in which the artificial key ID is defined in S:Employee, illustrates this.

Lines 4–6 in Algorithm 2 handle such cases.

Example 5. Consider the schema mapping μ_4 at the bottom of Figure 5(a), which has a similar effect to that of μ_2: to copy the name and email (variables n and c) of technical leaders of projects. The LAV alignments in Figure 5(b) now restrict leaders of projects to be female and experienced (i.e., leading at least three projects). The resulting verification statements produced by our framework to check the compatibility of μ_4 are:

$$\text{Variable } n : \mathit{Employee} \sqcap \mathit{Woman} \sqcap \mathit{SeniorLeader} \sqsubseteq \mathit{Leader} \tag{7}$$
$$\text{Variable } c : \mathit{Employee} \sqcap \mathit{Woman} \sqcap \mathit{SeniorLeader} \sqsubseteq \mathit{Leader} \tag{8}$$
$$\text{Variable } s : \mathit{Project} \sqcap \top \sqsubseteq \mathit{Project} \tag{9}$$

Thus, μ_4 is deemed compatible with the semantics of T.

4 Conclusion

We believe the framework described here holds promise as the basis for a practical tool to help in the verification of schema mappings. The main benefits of our

approach are as follows. First, unlike the manual verification of mappings, our method does not require the designer to be an expert on *both* the source and the target schemas. Second, our notion of semantic compatibility does not depend on problematic statistical assumptions as in other methods. Third, unlike with previous methods, our verification algorithm does not require data and does not rely on sampling. Fourth, considerable efforts have been made towards automatically identifying alignments, from which the annotations required in our method can be derived. We also posit that even if user intervention is necessary in this step, overall our approach is superior to the state-of-the-art, as providing an alignment from a specific schema into a general conceptual model should be easier than providing a mapping between two specific database schemas. The final advantage is economical: the efforts put into annotating a schema are amortized as the schema is integrated over and over again subsequently.

Related Work. Conceptual models have been used for many years as a means to allow database designers to convey the semantics of their schemas. Among the many formalisms proposed with this goal in mind, the family of Description Logics [6] stand out as a powerful and practical tool for describing conceptual models [6, Chapters 5 and 16]. The annotation of relational databases with conceptual descriptions has found renewed interest in many areas; e.g., providing better query interfaces through natural language processing [12]. Leveraging semantics for data integration is an active area of research [5, 4].

The semi-automatic verification of mappings has gained in importance recently. Besides Spicy [9], other tools have been proposed to help the schema mapping designer. Yan et al. [24] describe an approach to help users in understanding and refining schema mappings based on sampling. MUSE [3] helps users in resolving the interpretation of semantically ambiguous mappings. In [11], a schema mapping debugger is proposed to help the user understand the mappings with the aid of visual cues. Nevertheless, in all approaches above, the user is ultimately responsible for deciding the compatibility of the mappings.

Future Work. We see two immediate opportunities to extend this work. First, more expressive forms of schema alignments have been proposed, based on views [10] instead of relations, as considered here. Accommodating such alignments requires extending our notion of schema annotations and the way in which concepts are assigned to variables within a CQ as well. Also, it seems plausible to exploit the proof trees derived when checking the compatibility of mappings to (1) suggest ways of rectifying incompatible mappings or (2) *ranking* compatible mappings based on some notion of semantic distance.

References

1. Abiteboul et *al.* The lowell database research self-assessment. *Commun. ACM*, 48(5):111–118, 2005.
2. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

3. B. Alexe, L. Chiticariu, R. J. Miller, and W. C. Tan. MUSE: Mapping understanding and design by example. In *ICDE*, 2008.
4. Y. An, A. Borgida, R. Miller, and J. Mylopoulos. A semantic approach to discovering schema mapping expressions. In *ICDE 2007*, 2007.
5. Y. An, J. Mylopoulos, and A. Borgida. Building semantic mappings from databases to ontologies. In *AAAI*, 2006.
6. F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2nd edition, 2007.
7. T. Berners-Lee, J. Handler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, May 2001.
8. P. Bohannon, E. Elnahrawy, W. Fan, and M. Flaster. Putting context into schema matching. In *VLDB*, 2006.
9. A. Bonifati, G. Mecca, A. Pappalardo, S. Raunich, and G. Summa. Schema mapping verification: the spicy way. In *EDBT*, 2008.
10. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, R. Rosati, and M. Ruzzi. Data Integration through DL-LiteA Ontologies. In *Revised Selected Papers of the 3rd Int. Workshop on Semantics in Data and Knowledge Bases (SDKB 2008)*, volume 4925 of *Lecture Notes in Computer Science*. Springer, 2008.
11. L. Chiticariu and W. C. Tan. Debugging schema mappings with routes. In *VLDB*, 2006.
12. P. Cimiano, P. Haase, J. Heizmann, M. Mantel, and R. Studer. Towards portable natural language interfaces to knowledge bases - the case of the Orakel system. *Data Knowl. Eng.*, 65(2):325–354, 2008.
13. R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: semantics and query answering. *Theor. Comput. Sci.*, 336(1):89–124, 2005.
14. G. D. Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. On reconciling data exchange, data integration, and peer data management. In *PODS*, 2007.
15. R. J. Miller, M. A. Hernández, L. M. Haas, L.-L. Yan, C. T. H. Ho, R. Fagin, and L. Popa. The Clio project: Managing heterogeneity. *SIGMOD Record*, 30(1):78–83, 2001.
16. L. Popa, Y. Velegrakis, R. J. Miller, M. A. Hernández, and R. Fagin. Translating web data. In *VLDB*, 2002.
17. E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB J.*, 10(4):334–350, 2001.
18. P. Shvaiko and J. Euzenat. A survey of schema-based matching approaches. *J. Data Semantics IV*, 3730:146–171, 2005.
19. V. Sikka. Next generation data management in enterprise application platforms. In *VLDB*, 2006.
20. E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical owl-dl reasoner. *J. Web Sem.*, 5(2):51–53, 2007.
21. S. Tobies. The complexity of reasoning with cardinality restrictions and nominals in expressive description logics. *J. Artif. Intell. Res.*, 12:199–217, 2000.
22. S. Tobies. Complexity results and practical algorithms for logics in knowledge representation. *CoRR*, cs.LO/0106031, 2001.
23. D. Tsarkov and I. Horrocks. FaCT++ description logic reasoner: System description. In *IJCAR*, 2006.
24. L.-L. Yan, R. J. Miller, L. M. Haas, and R. Fagin. Data-driven understanding and refinement of schema mappings. In *SIGMOD*, 2001.

Optimal Reflection of Bidirectional View Updates using Information-Based Distance Measures

Stephen J. Hegner

Umeå University, Department of Computing Science
SE-901 87 Umeå, Sweden
hegner@cs.umu.se http://www.cs.umu.se/~hegner

Abstract. When a database view is to be updated, there are generally many choices for the new state of the main schema. One way of characterizing the best such choice is to minimize the distance between the old state of the main schema and the new one. In recent work, a means of representing such distance based upon semantics was forwarded in which the distance between two states is measured by the the difference of *information* between the two, with the information of a state defined as the set of sentences from a particular set which hold on that state. This approach proved to be highly useful in identifying optimal reflections of insertions and to a lesser extent deletions, provided that the reflections were themselves insertions or deletions. In this work, that investigation is extended to *bidirectional* view updates – those which involve both insertion and deletion. It is shown that the definition of distance must be crafted more carefully in such situations. Upon so doing, a result is obtained which provides update reflections which are information optimal for insertion and deletion optimal with respect to tuples but not necessarily information.

1 Introduction

The problem of supporting updates to databases through views has long been one which has challenged researchers. Roughly speaking, the approaches may be considered to lie in one of three groups. The first, as represented by [7, 20, 21, 4], focus upon using the relational algebra, together with null values, to identify optimal or at least acceptable solutions. These approaches often handle certain cases very well but lack the thread of a unifying theory. The second group has focused upon the constant-complement approach, first forwarded by Bancilhon and Spyratos [3] and subsequently developed in [14, 15, 22]. The approach provides an elegant theory within a limited scope, but the conservative nature of constant-complement-based strategies leaves uncovered many important situations. The third approach, developed largely within the logic-programming community, is based upon using distance measures to find solutions which change the database as little as possible [2, 12, 13]. The core idea behind these approaches is to minimize the "distance" between the old state of the main schema and

the new state which reflects the view update. Despite using logic as the tool for modelling, the distance is typically measured in a rather syntactic fashion, by minimizing the set of atoms, or even the number of atoms, which differ in the two databases. Such counting arguments fail to recapture that tuples are more than propositional atoms. For example, it is natural to expect the tuples $R(\mathrm{a}_1, \mathrm{b}_1, \mathrm{c}_1)$ and $R(\mathrm{a}_1, \mathrm{b}_1, \mathrm{c}_2)$ to be closer to each other than either is to $R(\mathrm{a}_3, \mathrm{b}_3, \mathrm{c}_3)$, but a measure which does not consider the inner structure of tuples cannot recapture this. To address this issue, more sophisticated measures of distance have been proposed [1] which are based upon (pseudo-)distance measures such as those proposed in [19] and [24]. These may then be extended to sets of tuples (i.e., databases) using aggregated measures, such as that of Eiter and Mannila [8]. Despite their obvious positive aspects, these measures nevertheless maintain a largely syntactic flavor.

In [17][1] and [16], an alternative approach is forwarded, in which a semantic notion of distance is employed, based upon the truth value of certain sentences rather than any syntactic characteristics of the atoms. The idea is to characterize a database state M by its information content $\mathsf{Info}\langle M, \Sigma\rangle$, the set of sentences in a set Σ which are true in M, and then to represent the distance between M and M' as the set of sentences in Σ whose truth values differ for the two states. The key issue is to choose Σ appropriately. It turns out that the most appropriate choice is $\mathsf{WFS}(\mathbf{D}, \exists\wedge+, K)$, the set of all existential, positive, and conjunctive sentences in the logic of the database schema $\mathbf{D}$ whose constant symbols are those of the set K consisting of all such symbols which occur in any of the current base state, the current view state, or the new view state. These sentences are just Boolean conjunctive queries [5]; i.e., conjunctive queries without free variables. A short example, adapted from that of [16, 1.3], will help illustrate the key ideas. For a more detailed presentation the reader is referred to that paper. Let $\mathbf{E}_0$ be the relational schema with relations $R[ABC]$ and $S[CD]$, constrained by the functional dependency (FD) $B \rightarrow C$ on $R[ABC]$ and the unary inclusion dependency (UIND) $R[C] \subseteq S[C]$, and let $M_{00} = \{R(\mathrm{a}_0, \mathrm{b}_0, \mathrm{c}_0), R(\mathrm{a}_1, \mathrm{b}_1, \mathrm{c}_1), S(\mathrm{c}_0, \mathrm{d}_0), S(\mathrm{c}_1, \mathrm{d}_1), S(\mathrm{c}_4, \mathrm{d}_4)\}$ be a database represented by ground atoms. Let $\Pi^{\mathbf{E}_0}_{R[AB]} = (R'[AB], \pi^{\mathbf{E}_0}_{R[AB]})$ be the view of $\mathbf{E}_0$ which projects $R[ABC]$ onto $R'[AB]$, dropping $S[A]$ entirely. Take M_{00} to be the initial state of schema $\mathbf{E}_0$; the corresponding view state is then $N_{00} = \{R'(\mathrm{a}_0, \mathrm{b}_0), R'(\mathrm{a}_1, \mathrm{b}_1)\}$. Now, suppose that the view update to insert $R'(\mathrm{a}_2, \mathrm{b}_2)$ is requested, so that the new view state will be $N_{01} = N_{00} \cup \{R'(\mathrm{a}_2, \mathrm{b}_2)\}$. The set of sentences over which information is measured is $\mathsf{WFS}(\mathbf{E}_0, \exists\wedge+, K_{00})$, with $K_{00} = \{\mathrm{a}_0, \mathrm{a}_1, \mathrm{a}_2, \mathrm{b}_0, \mathrm{b}_1, \mathrm{b}_2, \mathrm{c}_0, \mathrm{c}_1, \mathrm{d}_0, \mathrm{d}_1, \mathrm{c}_4, \mathrm{d}_4\}$ Any realization of this update must add $\varphi_{01} = (\exists x)(\exists y)(R(\mathrm{a}_2, \mathrm{b}_2, x) \wedge S(x, y))$ to the information content of M_{00}. A specific realization of this update must Skolemize the existentially quantified variables and add tuples such as those in $X_1 = \{(R(\mathrm{a}_2, \mathrm{b}_2, \bar{\mathrm{c}}_2), S(\bar{\mathrm{c}}_2, \bar{\mathrm{d}}_2)\}$ or $X_2 = \{(R(\mathrm{a}_2, \mathrm{b}_2, \bar{\mathrm{c}}_3), S(\bar{\mathrm{c}}_3, \bar{\mathrm{d}}_3)\}$, but since K_{00} does not contain any of the members of $\{\bar{\mathrm{c}}_2, \bar{\mathrm{d}}_2, \bar{\mathrm{c}}_3, \bar{\mathrm{d}}_3\}$, φ_{01} cannot distinguish between the solution which

[1] [17] contains some technical errors which have been corrected in the expanded version [16]. The reader is therefore referred to the latter paper.

adds X_1 from that which adds X_2. Thus, adding φ_{01} and its consequences in M_{00} to $\mathsf{Info}\langle M_{00}, \mathsf{WFS}(\mathbf{E}_0, \exists\wedge+, K_{00})\rangle$ characterizes the least added information necessary to realize the update, and so the distance from M_{00} to any Skolemized model of $M_{00} \cup \{\varphi_{01}\}$ must be least amongst all possibilities. Put another way, both $M_{00} \cup X_1$ and $M_{00} \cup X_2$ each have least distance from M_{00}; they differ only in the instantiation of variable by Skolem constants, and the logic cannot differentiate these instantiations. They are in a sense isomorphic solutions.

In [16], the focus is entirely upon unidirectional realizations of unidirectional update requests; that is, only view updates which are insertions and deletions are considered, and the reflection to the main schema must be of the same type as that of the request; insertions must reflected as insertions and deletions reflected as deletions. Suppose, now, that this restriction is dropped. The above solution is no longer optimal according to the definitions proposed. Indeed, the sentence $\varphi_{02} = \varphi_{01} \wedge S(\mathrm{c}_4, \mathrm{d}_4)$ is also added to $\mathsf{Info}\langle M_{00}, \mathsf{WFS}(\mathbf{E}_0, \exists\wedge+, K_{00})\rangle$ after the update, as are many others. This added information may be blocked by deleting $S(\mathrm{c}_4, \mathrm{d}_4)$, but that solution is not optimal either, since it deletes more information that the insert-only version. The addition of the sentence $S(\mathrm{c}_4, \mathrm{d}_4)$ to the inserted information is called a *collateral change*, because it is not part of the central update but rather a side effect of the way in which information change is measured. On the other hand, the sentence φ_{01} is essential to the update, and is called a *primary change*.

With an insertion which is to be reflected as an insertion, there is no problem with collateral changes. If φ_{01} is to be added, and $S(\mathrm{c}_4, \mathrm{d}_4)$ is already true, then it will still be true after the update. A similar observation applies to deletions. Of course, when the view update is an insertion, it is natural to require that it be reflected to the main schema as an insertion, and likewise for deletions. However, the problem of collateral change still remains for the class of view updates which involve both insertion and deletion.

The focus of this paper is to develop a formal framework for the representation of primary change which excludes collateral change, and to apply it to the characterization of the information-based distance between reflections, particularly in settings in which the update request involves both insertion and deletion. Notions of optimality based upon this measure are also developed.

2 The Underlying Context and Base Results

The underlying formalism for this paper is based heavily upon that which is developed in [16]. In this section, the essential terminology is reviewed, and a few useful extensions are presented.

Definition 2.1 (The relational model). All schemata are based upon a common *relational context* $\mathcal{D}$ which contains the attribute names and the constant names. Each domain has an infinite number of constants, but domains are allowed to overlap. Furthermore, there is a fixed *constant interpretation* $\mathcal{I}$ which enforces the unique name condition and which ensures that each domain

value is bound to a constant. A *relational schema* $\mathbf{D}$ consists of a set of relation symbols, each with an arity, together with a set $\mathsf{Constr}(\mathbf{D})$ of integrity constraints. $\mathsf{WFF}(\mathbf{D})$ denotes the set of all well-formed formulas in the language of $\mathbf{D}$ with equality, while $\mathsf{WFF}(\mathbf{D}, \exists\wedge+, K)$ denotes the subset of $\mathsf{WFF}(\mathbf{D})$ consisting of those formulas which are existential (no universal quantifiers), positive (no negation), and conjunctive (no disjunction), and which involve only those constant symbols in the set K. If K consists of all constants, this will be shortened to $\mathsf{WFF}(\mathbf{D}, \exists\wedge+)$. $\mathsf{WFS}(\mathbf{D})$, $\mathsf{WFS}(\mathbf{D}, \exists\wedge+)$, and $\mathsf{WFS}(\mathbf{D}, \exists\wedge+, K)$ denote the corresponding sets of sentences (with no free variables). Integrity constraints are special sentences in $\mathsf{WFS}(\mathbf{D})$; see Definition 2.5 below.

Databases are modelled as *finite* sets of ground atoms. $\mathsf{DB}(\mathbf{D})$ denotes the set of all databases of $\mathbf{D}$ without regard to constraints, while $\mathsf{LDB}(\mathbf{D})$, the *legal databases* of $\mathbf{D}$, consists of those which satisfy $\mathsf{Constr}(\mathbf{D})$. For $\Phi \subseteq \mathsf{WFS}(\mathbf{D})$, $\mathsf{AtMod}_{\mathcal{I}}(\Phi)$ denotes the set of all "models" of φ — the set of databases M which satisfy φ in the sense that $M \cup \{\neg\varphi \mid \varphi \in \Phi\}$ is not satisfiable. For $\varphi \in \mathsf{WFS}(\mathbf{D})$, $\mathsf{AtMod}_{\mathcal{I}}(\varphi)$ is shorthand for $\mathsf{AtMod}_{\mathcal{I}}(\{\varphi\})$. $\models_{\mathbf{D}}$ denotes semantic entailment within $\mathbf{D}$; $\Phi \models_{\mathbf{D}} \varphi$ holds iff $\mathsf{AtMod}_{\mathcal{I}}(\Phi \cup \mathsf{Constr}(\mathbf{D})) \subseteq \mathsf{AtMod}_{\mathcal{I}}(\varphi)$.

A database mapping $f : \mathbf{D}_1 \to \mathbf{D}_2$ is represented as a logical interpretation; i.e., in the relational calculus. Thus, for each relation symbol R of $\mathbf{D}_2$, there is a formula f^R in the language of $\mathbf{D}_1$ with free variables corresponding exactly to the attributes of R. It is of *class* $\exists\wedge+$ if each of its defining formulas is in $\mathsf{WFF}(\mathbf{D}_2, \exists\wedge+)$. Projection, selection, join, and intersection are all of class $\exists\wedge+$, while union and difference are not. For t an atom, the *substitution* of t into f is the result of mapping t to a formula in $\mathsf{WFS}(\mathbf{D}_1)$. For example, if f is the view mapping $\pi^{\mathbf{E}_0}_{R[AB]}$ of the example of Section 1, and t is $R'(\mathrm{a}_1, \mathrm{b}_1)$, then $f^{R'} = (\exists x_C)(R(x_A, x_B, x_C))$, and $\mathsf{Subst}(f, t) = (\exists x_C)(R(\mathrm{a}_1, \mathrm{b}_1, x_C))$.

A *view* of the schema $\mathbf{D}$ is a pair $\Gamma = (\mathbf{V}, \gamma)$ in which $\mathbf{V}$ is the *view schema* and $\gamma : \mathbf{D} \to \mathbf{V}$ is a database mapping which is surjective on the underlying databases. The view Γ is of class $\exists\wedge+$ iff its mapping γ has this property.

Notation 2.2 ($\Upsilon^{\mathbf{D}}$ and $\Upsilon^{\mathbf{D}}_K$). Because they occur so frequently, especially as arguments in even larger formulas, the sets $\mathsf{WFS}(\mathbf{D}, \exists\wedge+)$ and $\mathsf{WFS}(\mathbf{D}, \exists\wedge+, K)$ will often be abbreviated to $\Upsilon^{\mathbf{D}}$ and $\Upsilon^{\mathbf{D}}_K$, respectively.

Definition 2.3 (Information content). Let $\mathbf{D}$ be a database schema, let $K \subseteq \mathsf{ConstSym}(D)$, and let $M \in \mathsf{DB}(\mathbf{D})$. The *information content* of M relative to $\Upsilon^{\mathbf{D}}_K$ is the set of all sentences in $\Upsilon^{\mathbf{D}}_K$ which are true for M. In other words, $\mathsf{Info}\langle M, \Upsilon^{\mathbf{D}}_K\rangle = \{\varphi \in \Upsilon^{\mathbf{D}}_K \mid M \in \mathsf{AtMod}_{\mathcal{I}}(\varphi)\}$. Each $\varphi \in \Upsilon^{\mathbf{D}}_K$ defines a Boolean conjunctive query on M; the information content consists of just those queries which are true. Note that information content is monotone; if $M_1 \subseteq M_2$, then $\mathsf{Info}\langle M_1, \Upsilon^{\mathbf{D}}_K\rangle \subseteq \mathsf{Info}\langle M_2, \Upsilon^{\mathbf{D}}_K\rangle$.

Definition 2.4 (Armstrong models over $\mathsf{WFS}(\mathbf{D}, \exists\wedge+, K)$). Let $\Phi \subseteq \Psi \subseteq \Upsilon^{\mathbf{D}}_K$ for some set finite K of constant symbols. The *closure* of Φ in Ψ, denoted $\mathsf{Closure}\langle\Phi, \Psi\rangle$, is $\{\varphi \in \Psi \mid \Phi \models \varphi\}$. An *Armstrong model* for Φ relative to Ψ is a model which satisfies the constraints of $\mathsf{Closure}\langle\Phi, \Psi\rangle$ but no other constraints of

Ψ. Armstrong models have been studied extensively for database dependencies; see, for example, [9] and [11]. The construction within $\mathcal{Y}_K^{\mathrm{D}}$ is much simpler. For a finite set Φ, (almost) all that need be done is to rename variables so that all are distinct, remove the quantifiers, break the conjuncts into atoms, and then replace each variable with a distinct constant not occurring in K. For example, if $\Xi = \{R(\mathrm{a}_1, \mathrm{a}_2), R(\mathrm{a}_2, \mathrm{a}_3), (\exists x_1)(\exists x_2)(\exists x_3)(R(x_1, x_2) \wedge R(x_2, \mathrm{a}_3) \wedge R(\mathrm{a}_3, x_3))\}$. then $M_\Xi = \{R(\mathrm{a}_1, \mathrm{a}_2), R(\mathrm{a}_2, \mathrm{a}_3), R(\bar{\mathrm{a}}_1, \bar{\mathrm{a}}_2), R(\bar{\mathrm{a}}_2, \mathrm{a}_3), R(\mathrm{a}_3, \bar{\mathrm{a}}_3)\}$ is an Armstrong model of Ξ in $\mathcal{Y}_K^{\mathrm{D}}$, with $K = \{\mathrm{a}_1, \mathrm{a}_2, \mathrm{a}_3\}$.

In this work, the need is for *canonical* Armstrong models which are reduced in the sense that they contain no redundancies. M_Ξ is not reduced; indeed, $M_\Xi = \{R(\mathrm{a}_1, \mathrm{a}_2), R(\mathrm{a}_2, \mathrm{a}_3), R(\mathrm{a}_3, \bar{\mathrm{a}}_3)\}$ is also an Armstrong model of Ξ which is canonical because it contains no redundancies. For details of the construction, including an elaboration of this example, see see [16, 3.3-3.10]. As a notational convenience, Skolem constants, that is, constants which replace variables in the construction of Armstrong models, will always be written with a bar above the name to distinguish them from constant symbols which appear in the source formulas.

Definition 2.5 (Generalized Horn dependencies and canonical models). In this work, the constraints of database schemata will always be *generalized Horn dependencies* (GHDs), as described in [16, 3.15]. They are very similar to, but a bit more general than, the database dependencies of [9]. They are of the form below, but are not required to be typed, may involve constant symbols, and allow trivial left-hand sides.

$$(\forall x_1)(\forall x_2)\ldots(\forall x_m)((A_1 \wedge A_2 \wedge \ldots \wedge A_n) \Rightarrow (\exists y_1)(\exists y_2)\ldots(\exists y_r)(B_1 \wedge B_2 \wedge \ldots \wedge B_s))$$

They include all traditional dependencies, such as functional dependencies (FDs) and inclusion dependencies (IDs). They are further partitioned into equality-generating (EGHDs) and tuple-generating (TGHDs). Every $\varphi \in \mathsf{WFS}(\mathbf{D}, \exists\wedge+)$ is a GHD in which the left-hand side of the rule is trivially true.

Let $\mathbf{D}$ be a database schema, K a finite subset of $\mathsf{Constr}(\mathbf{D})$, and $\Phi \subseteq \mathcal{Y}_K^{\mathrm{D}}$. Define the *extended information* of Φ with respect to $\mathcal{Y}_K^{\mathrm{D}}$ to be $\mathsf{XInfo}_{\mathbf{D}}\langle \Phi, \mathcal{Y}_K^{\mathrm{D}} \rangle = \{\varphi \in \mathcal{Y}_K^{\mathrm{D}} \mid \Phi \models_{\mathbf{D}} \varphi\}$ if $\Phi \cup \mathsf{Constr}(\mathbf{D})$ is consistent, it is precisely the set of sentences which every legal database which also satisfies Φ must satisfy. Typically, Φ will be of the form $M_1 \cup \Psi$, with M_1 the current database state and Ψ the set of sentences which are to be inserted to effect the update. Using an approach similar to the traditional chase procedure [23, 8.6-8.8], it can be shown that if $\mathsf{XInfo}_{\mathbf{D}}\langle \Phi, \mathcal{Y}_K^{\mathrm{D}} \rangle$ is consistent, then it will always admit a model with least information provided the chase terminates [16, 3.20]. To ensure termination, it is sufficient that $\mathsf{Constr}(\mathbf{D})$ be *weakly acyclic* [10, Thm. 3.9]. Thus, under these conditions, an insertion to M_1 defined by a set $\Phi \subseteq \mathsf{WFS}(\mathbf{D}, \exists\wedge+, K)$ reflected from the view will always have a canonical least realization. Note that if Φ admits no model which is consistent with $\mathsf{Constr}(\mathbf{D})$, then $\mathsf{XInfo}_{\mathbf{D}}\langle \Phi, \mathcal{Y}_K^{\mathrm{D}} \rangle = \mathcal{Y}_K^{\mathrm{D}}$.

For $\Gamma = (\mathbf{V}, \gamma)$ a view of $\mathbf{D}$ of class $\exists\wedge+$ and $N \in \mathsf{DB}(\mathbf{D})$, $\mathsf{InfoLift}\langle N, \Gamma \rangle = \{\mathsf{Substf}\langle \gamma, t \rangle \mid t \in N\}$, the *lifting* of N to $\mathbf{D}$ along Γ. $\mathsf{XInfo}_{\mathbf{D}}\langle \mathsf{InfoLift}\langle N, \Gamma \rangle, \mathcal{Y}^{\mathrm{D}} \rangle$ is the least information which must hold in every $M \in \mathsf{LDB}(\mathbf{D})$ with $N \subseteq \gamma(M)$.

3 Update Requests and Generators

Notation 3.1. For the rest of this paper, unless stated explicitly to the contrary, $\mathbf{D}$ will be taken to be a relational schema with $\mathsf{Constr}(\mathbf{D})$ a set of GHDs whose TGHDs are weakly acyclic, and $\Gamma = (\mathbf{V}, \gamma)$ will be taken to be a view of $\mathbf{D}$ which is of class $\exists\wedge+$. Recall also that $\Upsilon^{\mathtt{D}}$ and $\Upsilon^{\mathtt{D}}_K$ are shorthand for $\mathsf{WFS}(\mathbf{D}, \exists\wedge+)$ and $\mathsf{WFS}(\mathbf{D}, \exists\wedge+, K)$, respectively.

Definition 3.2 (Updates, update requests, and realizations). An *update* on $\mathbf{D}$ is a pair $\mu = (M_1, M_2) \in \mathsf{LDB}(\mathbf{D}) \times \mathsf{LDB}(\mathbf{D})$. M_1 is the current state, and M_2 the new state. If $M_1 \subseteq M_2$, μ is called an *insertion*, and, dually, if $M_2 \subseteq M_1$, μ is called a *deletion*. Collectively, insertions and deletions are termed *unidirectional updates*. An update which is not unidirectional; i.e., which includes both the insertion and the deletion of tuples, is called *bidirectional*.

Let (N_1, N_2) be an update on the schema $\mathbf{V}$ of Γ. A *reflection* (or *translation*) of (N_1, N_2) along Γ is an update (M_1, M_2) on $\mathbf{D}$ with $M_i = \gamma(N_i)$ for $i \in \{1, 2\}$. In the view update problem, the current state M_1 of the main schema is known, as is the view update (N_1, N_2); it is the new state M_2 of the main schema which is to be determined. Thus, as an economy of notation, define an *update request* from Γ to $\mathbf{D}$ as a pair $u = (M_1, N_2)$ in which $M_1 \in \mathsf{LDB}(\mathbf{D})$ (the old state of the main schema) and $N_2 \in \mathsf{LDB}(\mathbf{V})$ (the new state of the view schema). The pair u is called an *insertion request* (resp. a *deletion request*, resp. a *bidirectional request*) precisely in the case that (N_1, N_2) has that property. A *realization* of (M_1, N_2) along Γ is a reflection (M_1, M_2) of (N_1, N_2).

The set $\mathfrak{C}_u$ consists of all constant symbols which occur in any of M_1, N_1, N_2, $\mathsf{Constr}(\mathbf{D})$, and the formulas of the view mapping γ. The information content of the result M_2 of a realization will usually be measured in $\Upsilon^{\mathtt{D}}_{\mathfrak{C}_u}$. This accounts for all constant symbols in the source databases, as well as in the constraints, but ignores any new constants introduced by Skolemization in the construction of a canonical Armstrong model.

The view Γ *reflects insertions* (resp. *reflects deletions*) if every insertion request (resp. deletion request) has a realization which is also an insertion (resp. deletion). Γ is *strongly monotonic* if it reflects both insertions and deletions. See [16, Sec. 5] for a discussion of these concepts and conditions which guarantee that they hold.

Definition 3.3 (Full update difference). It is important to recall the definition of update difference which was forwarded in [16], and which works well for unidirectional updates but not for bidirectional ones. The *positive (Δ^+), negative (Δ^-), and total (Δ) full update differences* of $\mu = (M_1, M_2) \in \mathsf{LDB}(\mathbf{D}) \times \mathsf{LDB}(\mathbf{D})$ with respect to $\Upsilon^{\mathtt{D}}_K$ are defined as $\Delta^+\langle\mu, \Upsilon^{\mathtt{D}}_K\rangle = \mathsf{Info}\langle M_2, \Upsilon^{\mathtt{D}}_K\rangle \setminus \mathsf{Info}\langle M_1, \Upsilon^{\mathtt{D}}_K\rangle$, $\Delta^-\langle\mu, \Upsilon^{\mathtt{D}}_K\rangle = \mathsf{Info}\langle M_1, \Upsilon^{\mathtt{D}}_K\rangle \setminus \mathsf{Info}\langle M_2, \Upsilon^{\mathtt{D}}_K\rangle$, and $\Delta\langle\mu, \Upsilon^{\mathtt{D}}_K\rangle = \Delta^+\langle\mu, \Upsilon^{\mathtt{D}}_K\rangle \cup \Delta^-\langle\mu, \Upsilon^{\mathtt{D}}_K\rangle$, respectively. Note that, given $\varphi \in \Delta\langle\mu, \Upsilon^{\mathtt{D}}_K\rangle$, it is always possible to determine whether $\varphi \in \Delta^+\langle\mu, \Upsilon^{\mathtt{D}}\rangle$ or $\varphi \in \Delta^-\langle\mu, \Upsilon^{\mathtt{D}}\rangle$ by checking whether or not $M_1 \in \mathsf{AtMod}_{\mathcal{I}}(\varphi)$.

As noted in the introduction, the problem with this definition for bidirectional updates is that the sets $\Delta^+\langle\mu, \Upsilon^{\mathtt{D}}\rangle$ and $\Delta^-\langle\mu, \Upsilon^{\mathtt{D}}\rangle$ contain compound

information, so that by adding elements to one, elements may be removed to the other. In the case of unidirectional updates, this is not an issue, since one of these sets will be empty. However, for bidirectional updates, both may be nonempty and this interference renders the measure less than completely useful.

Definition 3.4 (The semilattice $\mathcal{Y}^{\mathbf{D}}/\equiv_{\mathbf{D}}$). There is a natural order and a natural equivalence induced by $\mathsf{Constr}(\mathbf{D})$ on $\mathcal{Y}^{\mathbf{D}}$. To illustrate via example, in the schema $\mathbf{E}_0$ of Section 1, the inclusion dependency $R[C] \subseteq S[C]$ guarantees that whenever $(\exists x)(\exists y)(R(x, y, \mathrm{c}_0)$ is true in an arbitrary $M \in \mathsf{LDB}(\mathbf{E}_0)$, so too is $(\exists z)(S(\mathrm{c}_0, z))$. This is written as $(\exists x)(\exists y)(R(x, y, \mathrm{c}_0) \sqsubseteq_{\mathbf{E}_0} (\exists z)(S(\mathrm{a}_0, z))$. Similarly, the sentences $(\exists x)(\exists y)(R(x, y, \mathrm{c}_0)$ and $(\exists x)(\exists y)(R(x, y, \mathrm{c}_0) \wedge (\exists z)(S(\mathrm{a}_0, z))$ have identical truth values on all members of $\mathsf{LDB}(\mathbf{E}_0)$; this is written as $(\exists x)(\exists y)(R(x, y, \mathrm{c}_0) \equiv_{\mathbf{E}_0} (\exists x)(\exists y)(R(x, y, \mathrm{c}_0) \wedge (\exists z)(S(\mathrm{a}_0, z))$.

Formally, for the schema $\mathbf{D}$, define the preorder $\sqsubseteq_{\mathbf{D}}$ on $\mathcal{Y}^{\mathbf{D}}$ by $\varphi_1 \sqsubseteq_{\mathbf{D}} \varphi_2$ iff $\varphi_2 \models_{\mathbf{D}} \varphi_1$. In other words, $\varphi_1 \sqsubseteq_{\mathbf{D}} \varphi_2$ iff φ_2 is stronger than φ_1 on legal databases. Define the equivalence relation $\equiv_{\mathbf{D}}$ on $\mathcal{Y}^{\mathbf{D}}$ by $\varphi_1 \equiv_{\mathbf{D}} \varphi_2$ iff $\varphi_1 \sqsubseteq_{\mathbf{D}} \varphi_2 \sqsubseteq_{\mathbf{D}} \varphi_1$. Thus, $\equiv_{\mathbf{D}}$ identifies sentences which have identical truth values on all $M \in \mathsf{LDB}(\mathbf{D})$. The equivalence class of φ under $\equiv_{\mathbf{D}}$ is denoted by $[\varphi]_{\equiv_{\mathbf{D}}}$ or just $[\varphi]$, and the set of all such equivalence classes is $\mathcal{Y}^{\mathbf{D}}/\equiv_{\mathbf{D}}$. Upon grouping equivalent elements, a partial order is obtained. Specifically, define $[\varphi_1] \sqsubseteq_{\mathbf{D}} [\varphi_2]$ to hold iff $\varphi_1 \sqsubseteq_{\mathbf{D}} \varphi_2$. It is easy to see that the partial order $\sqsubseteq_{\mathbf{D}}$ on $\mathcal{Y}^{\mathbf{D}}/\equiv_{\mathbf{D}}$ defines a join-semilattice structure [6, Exer. 7.6] on $\sqsubseteq_{\mathbf{D}}$ with the join operation $\sqcup_{\mathbf{D}}$ given by $[\varphi_1] \sqcup_{\mathbf{D}} [\varphi_2] = [\varphi_1 \wedge \varphi_2]$.

Definition 3.5 (Ideals of $\mathcal{Y}^{\mathbf{D}}/\equiv_{\mathbf{D}}$). Sets of sentences which occur in this work, such as those of the form $\mathsf{Info}\langle M, \mathcal{Y}^{\mathbf{D}}_K\rangle$, are closed under implication within the context of the schema $\mathbf{D}$. The algebraic notion of an ideal of $\mathcal{Y}^{\mathbf{D}}/\equiv_{\mathbf{D}}$ provides a suitable form for the representation of such sets in a compact fashion. Specifically, an *ideal* [6, Exer. 7.6] of $\mathcal{Y}^{\mathbf{D}}/\equiv_{\mathbf{D}}$ is a subset which is closed downwards and under finite joins. More precisely, J is an ideal if (i) whenever $[\varphi_1] \in J$ and $[\varphi_2] \sqsubseteq_{\mathbf{D}} [\varphi_1]$, then $[\varphi_2] \in J$; and, (ii) whenever $[\varphi_1], [\varphi_2] \in J$, then $[\varphi_1] \sqcup_{\mathbf{D}} [\varphi_2] \in J$. Thus, ideals of $\mathcal{Y}^{\mathbf{D}}/\equiv_{\mathbf{D}}$ are closed under $\models_{\mathbf{D}}$ and conjunction. The set of all ideals of $\mathcal{Y}^{\mathbf{D}}/\equiv_{\mathbf{D}}$ is denoted $\mathsf{Ideals}(\mathcal{Y}^{\mathbf{D}}/\equiv_{\mathbf{D}})$.

Ideals may be described compactly by the elements which generate them. Specifically, for $\Phi \subseteq \mathcal{Y}^{\mathbf{D}}$, $\mathsf{Ideal}_{\mathbf{D}}(\Phi)$ is the smallest ideal of $\mathcal{Y}^{\mathbf{D}}/\equiv_{\mathbf{D}}$ containing $\{[\varphi] \mid \varphi \in \Phi\}$. Formally, it is the intersection of all such ideals. It is useful to have a notation for extracting the underlying sentences from an ideal. For $J \subseteq \mathcal{Y}^{\mathbf{D}}/\equiv_{\mathbf{D}}$, define $\|J\| = \{\varphi \mid [\varphi] \in J\}$.

Definition 3.6 (Specification of information change via ideals). To avoid the problem of collateral information change described in Section 1, the approach is to characterize $(\Delta^+\langle \mu, \mathcal{Y}^{\mathbf{D}}_{\mathfrak{C}_u}\rangle, \Delta^-\langle \mu, \mathcal{Y}^{\mathbf{D}}_{\mathfrak{C}_u}\rangle)$ as a pair of ideals. Formally, an *information-change specification* over $\mathbf{D}$ is given by an ordered pair $\langle G_+, G_{\not}\rangle \in \mathsf{Ideals}(\mathcal{Y}^{\mathbf{D}}/\equiv_{\mathbf{D}}) \times \mathsf{Ideals}(\mathcal{Y}^{\mathbf{D}}/\equiv_{\mathbf{D}})$. G_+ is the generator for the added information, and $G_{\not}$ is the generator for the information which is preserved.

To identify the information change associated with such a specification, let $M \in \mathsf{LDB}(\mathbf{D})$ and let $\langle G_+, G_{\not}\rangle$ be an information-change specification. The

new-state information for M induced by $\langle G_+, G_{\not=}\rangle$ relative to K is

$$\mathsf{NewInfo}_{\mathbf{D}}\langle M, \langle G_+, G_{\not=}\rangle, \Upsilon^{\mathrm{D}}_K\rangle = \mathsf{XInfo}_{\mathbf{D}}\langle \|G_+\| \cup \|G_{\not=}\|, \Upsilon^{\mathrm{D}}_K\rangle$$

Thus, the new information is just the sentences of G_+ and $G_{\not=}$, closed up under the constraints of $\mathbf{D}$. The set K of constants is determined from the update realization and not by $\langle G_+, G_{\not=}\rangle$ alone.

With this definition, it is possible to identify precisely the update which is induced by $\langle G_+, G_{\not=}\rangle$. Let $u = (M_1, N_2)$ be an update request from Γ to $\mathbf{D}$, and let $\mu = (M_1, M_2)$ be a realization of u. The pair $\langle G_+, G_{\not=}\rangle$ *generates* the realization μ for u if the following three conditions are satisfied.

(i) $G_+ \subseteq \mathsf{Ideal}_{\mathbf{D}}(\Delta^+\langle \mu, \Upsilon^{\mathrm{D}}_{\mathfrak{C}_u}\rangle)$.

(ii) $G_{\not=} = \mathsf{Ideal}_{\mathbf{D}}(\mathsf{Info}\langle M_1, \Upsilon^{\mathrm{D}}\rangle \setminus \Delta^-\langle \mu, \Upsilon^{\mathrm{D}}_{\mathfrak{C}_u}\rangle)$.

(iii) $\mathsf{NewInfo}_{\mathbf{D}}\langle M_1, \langle G_+, G_{\not=}\rangle, \Upsilon^{\mathrm{D}}_{\mathfrak{C}_u}\rangle = \mathsf{Info}\langle M_2, \Upsilon^{\mathrm{D}}_{\mathfrak{C}_u}\rangle$

There is no need for a subset representation in (ii) since collateral changes are limited to insertions. (Disjunction in the representation of information would be necessary for collateral changes in the deleted information to occur.)

This representation is similar in some ways to the (Insert, Retract) formalism of [2]. However, there are key differences. First, in [2] the elements of the update representation are ground atoms, whereas in the formalism of this paper they are sentences in $\Upsilon^{\mathrm{D}}_{\mathfrak{C}_u}$. Second, note that the generation of the deletion/retraction is given in complementary form — the set of sentences which are to be preserved, rather than the set of those which are to be deleted, is given. This a matter of convenience; it is much easier to represent the preserved information as an ideal than the deleted information as a filter, the dual of an ideal. Furthermore, special forms of updates, such as deletion optimality for tuples as presented in Definition 3.7 below, are much more succinctly specified via preservation.

Definition 3.7 (Optimality). The definitions of optimality are based upon subsumption in $\mathsf{Ideals}(\Upsilon^{\mathrm{D}}/\!\equiv_{\mathbf{D}})$. Let $u = (M_1, N_2)$ be an update request from Γ to $\mathbf{D}$, and let $\langle G_+, G_{\not=}\rangle \in \mathsf{Ideals}(\Upsilon^{\mathrm{D}}/\!\equiv_{\mathbf{D}}) \times \mathsf{Ideals}(\Upsilon^{\mathrm{D}}/\!\equiv_{\mathbf{D}})$ generate a realization for u. The ideal G_+ is *insertion optimal* for u if for any other $\langle G'_+, G'_{\not=}\rangle$ which defines a realization for u, $G_+ \sqsubseteq_{\mathbf{D}} G'_+$. Dually, $G_{\not=}$ is *deletion optimal* for u if for any other $\langle G'_+, G'_{\not=}\rangle$ which defines a realization for u, $G'_{\not=} \sqsubseteq_{\mathbf{D}} G_{\not=}$. Note the reversal of the inclusion for deletion optimality; a larger ideal preserves more and thus is to be preferred. Putting these together, $\langle G_+, G_{\not=}\rangle$ is *(fully) optimal* for u if both G_+ is insertion optimal and $G_{\not=}$ is deletion optimal for u.

Unfortunately, many update requests do not admit fully optimal solutions. There is, however, a restricted case which is often realizable. Define the *tuple ideals* of $\Upsilon^{\mathrm{D}}/\!\equiv_{\mathbf{D}}$ to be $\mathsf{Ideals}(\mathsf{DB}(\mathbf{D})) = \{\mathsf{Ideal}_{\mathbf{D}}(M) \mid M \in \mathsf{DB}(\mathbf{D})\}$. Thus, the tuple ideals are those which are generated by ground atoms; no quantifiers are allowed in the formulas. Call $G_{\not=}$ *deletion optimal for tuples* if $G_{\not=} \in \mathsf{Ideals}(\mathsf{DB}(\mathbf{D}))$ and for any other $\langle G'_+, G'_{\not=}\rangle$ which defines a realization for u and for which $G'_{\not=} \in \mathsf{Ideals}(\mathsf{DB}(\mathbf{D}))$, $G'_{\not=} \sqsubseteq_{\mathbf{D}} G_{\not=}$.

Examples 3.8. A few simple examples will help clarify these ideas. Let $\langle \mathbf{E}_0, \Pi^{\mathbf{E}_1}_{R[AB]}\rangle$ be the schema and view introduced in Section 1, with the current

state of the main schema $M_{00} = \{R(\mathrm{a}_0, \mathrm{b}_0, \mathrm{c}_0), R(\mathrm{a}_1, \mathrm{b}_1, \mathrm{c}_1), S(\mathrm{c}_0, \mathrm{d}_0), S(\mathrm{c}_1, \mathrm{d}_1), S(\mathrm{c}_4, \mathrm{d}_4)\}$ and the current state of the view $N_{00} = \{R'(\mathrm{a}_0, \mathrm{b}_0), R'(\mathrm{a}_1, \mathrm{b}_1)\}$. Suppose that the desired new state of the view is $N_{02} = \{R'(\mathrm{a}_0, \mathrm{b}_0), R'(\mathrm{a}_2, \mathrm{b}_2)\}$, so that the update request is $u_{00} = (M_{00}, N_{02})$

Consider the two generators $\langle G^{01}_+, G^{01}_\not\rangle$ and $\langle G^{02}_+, G^{02}_\not\rangle$ for u_{00}, with $G^{01}_+ = \mathsf{Ideal}_{\mathbf{E}_0}(\{(\exists z)(R(\mathrm{a}_2, \mathrm{b}_2, z))\})$, $G^{02}_+ = \mathsf{Ideal}_{\mathbf{E}_0}(\{(R(\mathrm{a}_2, \mathrm{b}_2, \mathrm{c}_1))\})$, $G^{01}_\not = \mathsf{Ideal}_{\mathbf{E}_{02}}(M_{00} \setminus \{R(\mathrm{a}_1, \mathrm{b}_1, \mathrm{c}_1)\})$, and $G^{02}_\not = \mathsf{Ideal}_{\mathbf{E}_0}(\mathsf{Info}\langle M_{00}, \Upsilon^{\mathbf{E}_0}\rangle \setminus \mathsf{Info}\langle\{(\exists z)(R(\mathrm{a}_1, \mathrm{b}_1, z))\}, \Upsilon^{\mathbf{E}_0}\rangle)$. The pair $\langle G^{01}_+, G^{01}_\not\rangle$ generates the update of M_{00} to $M_{01} = \{R(\mathrm{a}_0, \mathrm{b}_0, \mathrm{c}_0), R(\mathrm{a}_2, \mathrm{b}_2, \bar{\mathrm{c}}_2), S(\mathrm{c}_0, \mathrm{d}_0), S(\mathrm{c}_1, \mathrm{d}_1), S(\bar{\mathrm{c}}_2, \bar{\mathrm{d}}_2), S(\mathrm{c}_4, \mathrm{d}_4)\}$, while $\langle G^{02}_+, G^{02}_\not\rangle$ generates $M_{02} = \{R(\mathrm{a}_0, \mathrm{b}_0, \mathrm{c}_0), R(\mathrm{a}_2, \mathrm{b}_2, \mathrm{c}_1), S(\mathrm{c}_0, \mathrm{d}_0), S(\mathrm{c}_1, \mathrm{d}_1), S(\mathrm{c}_4, \mathrm{d}_4)\}$. Roughly, $\langle G^{01}_+, G^{01}_\not\rangle$ corresponds to deleting $R'(\mathrm{a}_0, \mathrm{b}_0)$ and inserting $R'(\mathrm{a}_2, \mathrm{b}_2)$, while $\langle G^{02}_+, G^{02}_\not\rangle$ corresponds to the replacements $\mathrm{a}_1 \mapsto \mathrm{a}_2$ and $\mathrm{b}_1 \mapsto \mathrm{b}_2$ in $R'(\mathrm{a}_1, \mathrm{b}_1)$. The pair $\langle G^{01}_+, G^{01}_\not\rangle$ is insertion optimal; this will be proven more generally in Proposition 4.3, but in this case that fact is easily seen by inspection. It is also deletion optimal for tuples, as will be established more generally in Theorem 4.6. However, it is not deletion optimal in general, since $\langle G^{02}_+, G^{02}_\not\rangle$ deletes less information. More specifically, the sentence $(\exists x)(\exists y)(R(x, y, \mathrm{c}_1) \wedge S(\mathrm{c}_1, \mathrm{c}_1))$ is preserved with $\langle G^{02}_+, G^{02}_\not\rangle$ but not $\langle G^{01}_+, G^{01}_\not\rangle$. On the other hand, more information is inserted with $\langle G^{02}_+, G^{02}_\not\rangle$ as well, since $R(\mathrm{a}_2, \mathrm{b}_2, \mathrm{c}_1) \models_{\mathbf{E}_0} (\exists z)(R(\mathrm{a}_2, \mathrm{b}_2, z))$, but not conversely. It is easy to see by inspection that $\langle G^{02}_+, G^{02}_\not\rangle$ is deletion optimal. Since $\langle G^{01}_+, G^{01}_\not\rangle$ is insertion optimal, no solution which is both insertion and deletion optimal can exist. In Theorem 4.6, it will be shown that under conditions satisfied by $\langle \mathbf{E}_0, \Pi^{\mathbf{E}_0}_{R[AB]}\rangle$, generators such as $\langle G^{01}_+, G^{01}_\not\rangle$ which are insertion optimal without restriction as well as deletion optimal for tuples always exist.

It is not always the case that deletion-optimal generators exist. For example, define $N_{03} = \{R'(\mathrm{a}_2, \mathrm{b}_2)\}$, with the update request $u_{03} = (M_{00}, N_{03})$. There is an insertion-optimal generator $\langle G^{03}_+, G^{03}_\not\rangle$, given by $G^{03}_+ = \mathsf{Ideal}_{\mathbf{E}_0}(\{(\exists z)(R(\mathrm{a}_2, \mathrm{b}_2, z))\})$ and $G^{03}_\not = \mathsf{Ideal}_{\mathbf{E}_0}(M_{03} \setminus \{R(\mathrm{a}_0, \mathrm{b}_0, \mathrm{c}_0), R(\mathrm{a}_1, \mathrm{b}_1, \mathrm{c}_1)\})$ with the resulting state of the form $M_{03} = \{R(\mathrm{a}_2, \mathrm{b}_2, \bar{\mathrm{c}}_2), S(\mathrm{c}_0, \mathrm{d}_0), S(\mathrm{c}_1, \mathrm{d}_1), S(\bar{\mathrm{c}}_2, \bar{\mathrm{d}}_2), S(\mathrm{c}_4, \mathrm{d}_4)\}$. However, there is no deletion-optimal generator. Indeed, consider the two states $M_{03} = \{R(\mathrm{a}_2, \mathrm{b}_2, \mathrm{c}_0), S(\mathrm{c}_0, \mathrm{d}_0), S(\mathrm{c}_1, \mathrm{d}_1), S(\mathrm{c}_2, \mathrm{d}_2), S(\mathrm{c}_4, \mathrm{d}_4)\}$ and $M'_{03} = \{R(\mathrm{a}_2, \mathrm{b}_2, \mathrm{c}_1), S(\mathrm{c}_0, \mathrm{d}_0), S(\mathrm{c}_1, \mathrm{d}_1), S(\mathrm{c}_2, \mathrm{d}_2), S(\mathrm{c}_4, \mathrm{d}_4)\}$, which are generated by $\langle G^{04}_+, G^{04}_\not\rangle$ and $\langle G^{04'}_+, G^{04'}_\not\rangle$ respectively, with $G^{04}_+ = \mathsf{Ideal}_{\mathbf{E}_0}(\{R(\mathrm{a}_2, \mathrm{b}_2, \mathrm{c}_0)\})$, $G^{04'}_+ = \mathsf{Ideal}_{\mathbf{E}_0}(\{R(\mathrm{a}_2, \mathrm{b}_2, \mathrm{c}_1))\})$, $G^{04}_\not = \mathsf{Ideal}_{\mathbf{E}_0}(\mathsf{Info}\langle M_{03}, \Upsilon^{\mathbf{E}_0}\rangle \setminus \mathsf{Info}\langle\{(\exists z)(R(\mathrm{a}_0, \mathrm{b}_0, z)), R(\mathrm{a}_1, \mathrm{b}_1, \mathrm{c}_1)\}, \Upsilon^{\mathbf{E}_0}\rangle)$ and $G^{04'}_\not = \mathsf{Ideal}_{\mathbf{E}_0}(\mathsf{Info}\langle M_{03}, \Upsilon^{\mathbf{E}_0}\rangle \setminus \mathsf{Info}\langle\{(\exists z)(R(\mathrm{a}_1, \mathrm{b}_1, z)), R(\mathrm{a}_0, \mathrm{b}_0, \mathrm{c}_0)\}, \Upsilon^{\mathbf{E}_0}\rangle)$. There is no way to include both $(\exists x)(\exists y)(R(x, y, \mathrm{b}_0))$ and $(\exists x)(\exists y)(R(x, y, \mathrm{b}_1))$ in a solution without violating the FD $B \to C$, since y must be bound b_2 in every R-tuple of a solution.

Example 3.9 (Full optimality). Optimal solutions do exist in certain situations, and it is instructive to illustrate one of them. Let $\mathbf{E}_1$ have the single relational symbol $R[ABC]$, constrained by the join dependency $\bowtie [AB, BC]$.

Define the view $\Pi^{\mathbf{E}_1}_{R[AB]} = (R'[AB], \pi^{\mathbf{E}_1}_{R[AB]})$ to be that which projects $R[ABC]$ onto $R'[AB]$. Let $M_{10} = \{R(\mathrm{a}_0, \mathrm{b}_0, \mathrm{c}_0), R(\mathrm{a}_1, \mathrm{b}_1, \mathrm{c}_1)\} \in \mathsf{LDB}(\mathbf{E}_1)$; the corresponding view state is then $N_{10} = \{R'(\mathrm{a}_0, \mathrm{b}_0), R'(\mathrm{a}_1, \mathrm{b}_1)\}$. Consider the update request $u_{10} = (M_{10}, N_{11})$ with $N_{11} = \{R'(\mathrm{a}_0, \mathrm{b}_0), R'(\mathrm{a}_2, \mathrm{b}_1)\}$ and the solution (M_{10}, M_{11}) with $M_{11} = \{R(\mathrm{a}_0, \mathrm{b}_0, \mathrm{c}_0), R(\mathrm{a}_2, \mathrm{b}_1, \mathrm{c}_1)\}$. This solution is optimal; indeed, it is generated by $\langle G^{10}_+, G^{10}_{\not\vdash}\rangle$ with $G^{10}_+ = \{(\exists z)(R(\mathrm{a}_2, \mathrm{b}_1, z))\}$ and $G^{10}_{\not\vdash} = \mathsf{Ideal}_{\mathbf{E}_0}(\mathsf{Info}\langle M_{10}, \Upsilon^{\mathbf{E}_1}\rangle \setminus \mathsf{Info}\langle\{(\exists z)(R(\mathrm{a}_1, \mathrm{b}_1, z))\}, \Upsilon^{\mathbf{E}_1}\rangle)$, and it is easy to see that every solution must insert $(\exists z)(R(\mathrm{a}_2, \mathrm{b}_1, z))$ and delete $(\exists z)(R(\mathrm{a}_1, \mathrm{b}_1, z))$. This is an example of constant-complement update [3], [16], addressed further in Discussion 4.7.

4 Optimal Realization of Bi-Directional Update Requests

Example 4.1 (Motivating example). For any schema $\mathbf{D}$ satisfying the conditions spelled out in Notation 3.1, and any update request $u = (M_1, N_2)$, a sure way to obtain an insertion-optimal solution is to forget entirely what is M_1 and just use a canonical Armstrong model of $\mathsf{InfoLift}\langle N_2, \Gamma\rangle$. This is best illustrated by example; consider again the pair $\langle \mathbf{E}_0, \Pi^{\mathbf{E}_1}_{R[AB]}\rangle$ and the view state $N_{02} = \{R'(\mathrm{a}_0, \mathrm{b}_0), R'(\mathrm{a}_2, \mathrm{b}_2)\}$ of Examples 3.8. $\mathsf{InfoLift}\langle N_{02}, \Pi^{\mathbf{E}_0}_{R'[AB]}\rangle = \{(\exists z)(R(\mathrm{a}_0, \mathrm{b}_0, z)), (\exists z)(R(\mathrm{a}_2, \mathrm{b}_2, z))\}$, whose canonical Armstrong models in $\mathbf{E}_0$ are of the form $M_{02} = \{R(\mathrm{a}_0, \mathrm{b}_0, \bar{\mathrm{c}}_0), R(\mathrm{a}_2, \mathrm{b}_2, \bar{\mathrm{c}}_2)\}$. To make sure that the constant symbols $\bar{\mathrm{c}}_0$ and $\bar{\mathrm{c}}_2$ are not in $\mathfrak{C}_u$, with $u = (M, N_{02})$, it is necessary to know what the constant symbols of M are, but the construction of $\mathsf{InfoLift}\langle N_{02}, \Pi^{\mathbf{E}_0}_{R'[AB]}\rangle$ itself does not depend upon M or its constant symbols.

The generating pair may be written as $G_{04} = \langle G^{04}_+, G^{04}_{\not\vdash}\rangle = \langle \mathsf{Ideal}_{\mathbf{E}_0}(\{(\exists z)(R(\mathrm{a}_0, \mathrm{b}_0, z)), (\exists z)(R(\mathrm{a}_2, \mathrm{b}_2, z))\}), \emptyset\rangle$, but this need not be an optimal representation. For example, if the initial state is $M_{00} = \{R(\mathrm{a}_0, \mathrm{b}_0, \mathrm{c}_0), R(\mathrm{a}_1, \mathrm{b}_1, \mathrm{c}_1), S(\mathrm{c}_0, \mathrm{d}_0), S(\mathrm{c}_1, \mathrm{d}_1), S(\mathrm{c}_4, \mathrm{d}_4)\}$, then $G^{04}_{\not\vdash}$ specifies the deletion of $(\exists z)(R(\mathrm{a}_0, \mathrm{b}_0, x))$ while G^{04}_+ then mandates its reinsertion. The optimal generator $G'_{04} = \langle G^{04\prime}_+, G^{04\prime}_{\not\vdash}\rangle = \langle \mathsf{Ideal}_{\mathbf{E}_0}(\{(\exists z)(R(\mathrm{a}_2, \mathrm{b}_2, z))\}), \mathsf{Ideal}_{\mathbf{E}_0}(\mathsf{XInfo}_{\mathbf{E}_0}\langle M_{00}, \Upsilon^{\mathbf{E}_0}\rangle \setminus \mathsf{XInfo}_{\mathbf{E}_0}\langle\{(\exists z)(R(\mathrm{a}_0, \mathrm{b}_0, z))\}, \Upsilon^{\mathbf{E}_0}\rangle)\rangle$ avoids this problem. Returning to the general situation of $u = (M_1, N_2)$ on Γ and a realization $G = \langle G_+, G_{\not\vdash}\rangle$, the information in $\mathsf{Info}\langle M_1, \Upsilon^{\mathbf{D}}\rangle$ which is also in $\mathsf{InfoLift}\langle N_2, \Gamma\rangle$ should be specified in $G_{\not\vdash}$, not in G_+. The formalization of these ideas follows.

Definition 4.2. Let $u = (M_1, N_2)$ be an update request from Γ to $\mathbf{D}$. Define the *least insertion-optimal realization* $G^{\langle u+\rangle} = \langle G^{u+}_+, G^{u+}_{\not\vdash}\rangle$ of u as

$$G^{\langle u+\rangle}_+ = \mathsf{Ideal}_{\mathbf{D}}(\mathsf{InfoLift}\langle N_2, \Gamma\rangle \setminus \mathsf{Info}\langle M_1, \Upsilon^{\mathbf{D}}\rangle)$$
$$G^{\langle u+\rangle}_{\not\vdash} = \mathsf{Ideal}_{\mathbf{D}}(\mathsf{Info}\langle M_1, \Upsilon^{\mathbf{D}}\rangle \cap \mathsf{InfoLift}\langle N_2, \Gamma\rangle)$$

Proposition 4.3. *Let $u = (M_1, N_2)$ be an update request from Γ to $\mathbf{D}$. Assume further that Γ reflects deletions. Then $G^{\langle u+\rangle}$ generates an insertion-optimal realization $\mu = (M_1, M_2)$ for u, with M_2 any canonical Armstrong model for*

$\mathsf{InfoLift}\langle N_2, \Gamma\rangle$. *The update μ has the further property that for any other realization $\mu' = (M_1, M_2')$, $\mathsf{Info}\langle M_2, \Upsilon^{\mathrm{D}}\rangle \subseteq \mathsf{Info}\langle M_2', \Upsilon^{\mathrm{D}}\rangle$.*

Proof. It is clear that the construction produces the least information which a state $M_2 \in \mathsf{LDB}(\mathbf{D})$ for which $\gamma(M_2) = N_2$ must possess. The only concern is that $\gamma(M_2)$ may contain phantom tuples; that is, tuples which involve constant symbols resulting from the process of Skolemizing existentially quantified variables in the conversion from the information set to a canonical Armstrong model. The condition that Γ reflect deletions ensures that such tuples are impossible. For a discussion of this phenomenon, with examples, and a proof of the result that reflection of deletions prevents phantom tuples, consult [16, 4.8-4.11]. □

Definition 4.4 (Unit-head pairs). Tuple generating dependencies with more than one atom on the left-hand side create problems for deletions. If a rule of the form $A_1 \wedge A_2 \Rightarrow B$ holds, and B is to be deleted, then there is a choice of whether to delete A_1 or A_2, and so no least deletion exists. To obtain useful optimization results in the context of deletions, a restricted form of schema-view pair called a unit-head pair [16, 6.10] is appropriate. Since the database mapping γ of the view Γ also introduces constraints, it is first necessary to construct the *combined schema* $\mathsf{CombSch}\langle\mathbf{D}, \Gamma\rangle$, in which the main schema is augmented with the relation symbols of the view. For $\mathbf{E}_0$ introduced in Section 1, the relation symbol $R'[AB]$, as well as the constraint $(\forall x)(\forall y)(R'(x, y) \Leftrightarrow (\exists z)(R(x, y, z)))$ are added to the main schema. This constraint decomposes into the TGHDs $(\forall x)(\forall y)(R'(x, y) \Rightarrow (\exists z)(R(x, y, z)))$ and $(\forall x)(\forall y)(\forall z)(R(x, y, z) \Rightarrow R'(x, y))$. Now, call the pair $\langle\mathbf{D}, \Gamma\rangle$ *unit head* if each of the TGHDs of $\mathsf{CombSch}\langle\mathbf{D}, \Gamma\rangle$ is *unit head*; i.e., has at most one atom on the left-hand side. EGHDs are allowed without restriction, as are elements of Υ^{D} and so-called mutual-exclusion dependencies of the form $A_1 \wedge \ldots \wedge A_k \Rightarrow \bot$. The pair $\langle\mathbf{E}_0, \Pi^{\mathbf{E}_0}_{R[AB]}\rangle$ of Section 1 and Examples 3.8 is unit head, while the pair $\langle\mathbf{E}_1, \Pi^{\mathbf{E}_1}_{R[AB]}\rangle$ of Example 3.9 is not.

Definition 4.5 (Reflection of general-source insertions). The notions of reflecting insertions and reflecting deletions play a central rôle in the characterization of views which support the reflection of unidirectional update requests in an optimal fashion. For bidirectional updates, a stronger version of insertion reflection is necessary, in which the source need not be a legal database but rather only a database which may be extended to a legal one. Formally, a *general-source insertion specification* is a pair $u = (M_1, N_2) \in \mathsf{DB}(\mathbf{D}) \times \mathsf{LDB}(\mathbf{V})$ with the property that (i) $\gamma(M_1) \subseteq N_2$ and (ii) there is some $M_1' \in \mathsf{LDB}(\mathbf{D})$ with $M_1 \subseteq M_1'$. A *realization* of u is a pair $(M_1, M_2) \in \mathsf{DB}(\mathbf{D}) \times \mathsf{LDB}(\mathbf{D})$ with the property that $M_1 \subseteq M_2$ and $\gamma(M_2) = N_2$. Say that Γ *reflects general-source insertions* if every general-source insertion specification admits a realization.

It is not known (at least not to the author) whether this condition is strictly stronger than insertion reflection [16, 4.13], which requires in addition that $M_1 \in \mathsf{LDB}(\mathbf{D})$ in the above. However, it is a simple exercise to show that it is satisfied by schemata which are constrained by FDs and UINDs (unary inclusion dependencies) with Γ is both FD-complete and UIND-complete. For a more complete presentation, see [16, Sec. 5].

Theorem 4.6. *Let $\langle \mathbf{D}, \Gamma \rangle$ be a unit-head pair which reflects deletions and general-source insertions, and let $u = (M_1, N_2)$ be an update request from Γ to $\mathbf{D}$. Define $P = \{t \in M_1 \mid \gamma(\{t\}) \subseteq N_2\}$. Then $\langle G_+^{\langle u+ \rangle}, \mathsf{Ideal}_{\mathbf{D}}(P) \rangle$ is both insertion optimal without restriction and deletion optimal for tuples for u.*

Proof. First note that because $\langle \mathbf{D}, \Gamma \rangle$ is unit-head, every interpretation formula γ^R of the view morphism γ consists of a single atom. Thus, γ is defined entirely by its action upon tuples; i.e., $\gamma(M) = \{\gamma(\{t\}) \mid t \in M\}$. Therefore, P as defined above is clearly the largest subset of M with the property that $\gamma(P) \subseteq N_2$; no larger subset $P' \subseteq M$ can possibly have the property that $\langle G_+^{\langle u+ \rangle}, \mathsf{Ideal}_{\mathbf{D}}(P') \rangle$ generates a realization for u. On the other hand, since Γ reflects general-source deletions and $\gamma(P) \subseteq N_2$, there must be an $M_2' \in \mathsf{LDB}(\mathbf{D})$ such that $P \subseteq M_2'$ and $\gamma(M_2') = N_2$. Since $\mathsf{XInfo}_{\mathbf{D}} \langle P \cup \mathsf{InfoLift} \langle N_2, \Gamma \rangle, \Upsilon^{\mathbf{D}} \rangle$ is the least information which any $M \in \mathsf{LDB}(\mathbf{D})$ which both contains P and satisfies $\mathsf{InfoLift} \langle N_2, \Gamma \rangle$ must have, $\mathsf{XInfo}_{\mathbf{D}} \langle P \cup \mathsf{InfoLift} \langle N_2, \Gamma \rangle, \Upsilon^{\mathbf{D}} \rangle \subseteq \mathsf{Info} \langle M_2', \Upsilon^{\mathbf{D}}_{\mathfrak{C}_u} \rangle$. Thus, $\mathsf{XInfo}_{\mathbf{D}} \langle P \cup \mathsf{InfoLift} \langle N_2, \Gamma \rangle, \Upsilon^{\mathbf{D}} \rangle$ is consistent and so admits a canonical Armstrong model M_2. This model must furthermore have the property that $\gamma(M_2) = N_2$. Indeed, if $\gamma(M_2)$ were to contain tuples not in N_2, they could be deleted using the update specification (M_2, N_2) and the fact that Γ reflects deletions. See [16, 4.10] for details surrounding this argument. That $\langle G_+^{\langle u+ \rangle}, \mathsf{Ideal}_{\mathbf{D}}(P) \rangle$ is insertion optimal follows from Proposition 4.3. □

See Examples 3.8 for examples of the above construction.

Discussion 4.7 (The optimality of constant-complement solutions). In [18], constant-complement update strategies ([3], [14]) are investigated the context of information-based distance measures, with the main result [18, 4.23] establishing that all such strategies based upon so-called semantically bijective decompositions are optimal. While this result is valid, the proof is in error in that it fails to take into account collateral information changes and works implicitly with the assumption that minimization of the size of a generator suffices. Fortunately, using the framework of the current paper, this use of generators may easily be made explicit and the repair of the proof of [18, 4.23] is almost trivial. The correct proof will appear in a revised version. *Thus, broadly stated, constant-complement update strategies are fully optimal.*

Example 4.8 (The limitations of semantic distance measures). Let $\mathbf{E}_2$ be the schema with two unary relation symbols $R[A]$ and $S[A]$, with no constraints, and let $\Pi^{\mathbf{E}_2}_{R[A]} = (R[A], \pi^{\mathbf{E}_2}_{R[A]})$ be the view of $\mathbf{E}_2$ which retains $R[A]$ entirely while discarding $S[A]$ completely. It is clear that any view update should keep $S[A]$ fixed; this is a simple example of the constant-complement strategy. Now, let $\mathbf{E}_3$ be identical to $\mathbf{E}_2$ save that a new relation symbol $T[A]$ is introduced with the constraint $(\forall x)(R(x) \wedge S(x) \Leftrightarrow T(x))$. The view $\Pi^{\mathbf{E}_3}_{R[A]} = (R[A], \pi^{\mathbf{E}_3}_{R[A]})$ is the same as $\Pi^{\mathbf{E}_2}_{R[A]}$. The two schemata $\mathbf{E}_2$ and $\mathbf{E}_3$ are logically equivalent, since $T[A]$ is defined completely in terms of $R[A]$ and $S[A]$. The isomorphism connecting them is even of class $\exists \wedge +$. Yet it is not clear that the optimal update

strategies should be the same. If the state of $\mathbf{E}_3$ is $M_{31} = \{R(\mathrm{a}_0), S(\mathrm{a}_1)\}$, and the desired new view state is $N_{32} = \{R(\mathrm{a}_0), R(\mathrm{a}_1)\}$, then the optimal strategy, as defined by the theory of this paper, is to insert $R(\mathrm{a}_1)$ and keep $S[A]$ constant, which thus triggers an insertion of $T(\mathrm{a}_1)$. However, by deleting $S(\mathrm{a}_1)$, this insertion into $T[A]$ could be avoided.

In light of this example, one might argue that perhaps G_+ of an update generator $\langle G_+, G_{\not=}\rangle$ should only insert things which are not covered by inference; that is, to define $\mathsf{NewInfo}_{\mathbf{D}}\langle M, \langle G_+, G_{\not=}\rangle, \Upsilon_K^{\mathrm{D}}\rangle = \mathsf{XInfo}_{\mathbf{D}}\langle \|G_+\|, \Upsilon_K^{\mathrm{D}}\rangle \cup \mathsf{XInfo}_{\mathbf{D}}\langle \|G_{\not=}\|, \Upsilon_K^{\mathrm{D}}\rangle$; then $\{R(\mathrm{a}_1), R(\mathrm{a}_2), S(\mathrm{a}_2), T(\mathrm{a}_2)\}$ would no longer be preferred to $\{R(\mathrm{a}_1), R(\mathrm{a}_2)\}$ as a solution. Unfortunately, this strategy breaks the optimality of a constant-complement update defined by a join, such as in $\mathbf{E}_1$ of Example 3.9. Additional research is necessary to identify ways to retain the semantic nature of the proposed distance measures while respecting the kind of syntactic constructions illustrated in $\mathbf{E}_3$.

5 Conclusions and Further Directions

An approach to characterizing the distance between database states when both insertion and deletion are involved has been presented. In contrast to syntax-based approaches, it attempts to quantify the difference in meaning and thus adds a dimension which other distance measures lack.

On the other hand, as illustrated in Example 4.8, a semantics-based approach can sometimes produce questionable results (as can a syntax-based approach, of course). Therefore, an important next step in this research is to investigate ways to integrate both syntactic- and semantic-based measures to retain the best aspects of each. Methods for computing the distance between two states, at least for restricted classes of schemata, are also essential if this type of approach is to achieve practical utility.

References

1. O. Arieli, M. Denecker, and M. Bruynooghe. Distance semantics for database repair. *Ann. Math. Artif. Intell.*, 50(3-4):389–415, 2007.
2. O. Arieli, M. Denecker, B. V. Nuffelen, and M. Bruynooghe. Computational methods for database repair by signed formulae. *Ann. Math. Artif. Intell.*, 46(1-2):4–37, 2006.
3. F. Bancilhon and N. Spyratos. Update semantics of relational views. *ACM Trans. Database Systems*, 6:557–575, 1981.
4. F. Bentayeb and D. Laurent. Inversion de l'algèbre relationnelle et mises à jour. Technical Report 97-9, Université d'Orléans, LIFO, 1997.
5. A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proceedings of the Ninth Annual ACM Symposium on Theory of Computing, 2-4 May 1977, Boulder, Colorado*, pages 77–90, 1977.
6. B. A. Davey and H. A. Priestly. *Introduction to Lattices and Order*. Cambridge University Press, second edition, 2002.

7. U. Dayal and P. A. Bernstein. On the correct translation of update operations on relational views. *ACM Trans. Database Systems*, 8(3):381–416, 1982.
8. T. Eiter and H. Mannila. Distance measures for point sets and their computation. *Acta Inf.*, 34(2):109–133, 1997.
9. R. Fagin. Horn clauses and database dependencies. *J. Assoc. Comp. Mach.*, 29(4):952–985, 1982.
10. R. Fagin, P. G. Kolaitis, R. J. Miller, and L. Popa. Data exchange: Semantics and query answering. *Theoret. Comput. Sci.*, 336:89–124, 2005.
11. R. Fagin and M. Y. Vardi. Armstrong databases for functional and inclusion dependencies. *Info. Process. Lett.*, 16:13–19, 1983.
12. J. A. Fernández, J. Grant, and J. Minker. Model theoretic approach to view updates in deductive databases. *J. Automated Reasoning*, 17(2):171–197, 1996.
13. S. Greco, C. Sirangelo, I. Trubitsyna, and E. Zumpano. Preferred repairs for inconsistent databases. In *7th International Database Engineering and Applications Symposium (IDEAS 2003), 16-18 July 2003, Hong Kong, China*, pages 202–211. IEEE Computer Society, 2003.
14. S. J. Hegner. An order-based theory of updates for closed database views. *Ann. Math. Art. Intell.*, 40:63–125, 2004.
15. S. J. Hegner. The complexity of embedded axiomatization for a class of closed database views. *Ann. Math. Art. Intell.*, 46:38–97, 2006.
16. S. J. Hegner. Information-based distance measures and the canonical reflection of view updates. Technical Report 0805, Institut für Informatik, Christian-Albrechts-Universität zu Kiel, October 2008. Also available on the web site of the author.
17. S. J. Hegner. Information-optimal reflections of view updates on relational database schemata. In S. Hartmann and G. Kern-Isberner, editors, *Foundations of Information and Knowledge Systems: Fifth International Symposium, FoIKS 2008, Pisa, Italy, February 11-15, 2008, Proceedings*, volume 4932 of *Lecture Notes in Computer Science*, pages 112–131. Springer-Verlag, 2008.
18. S. J. Hegner. Semantic bijectivity and the uniqueness of constant-complement updates in the relatiional context. In K.-D. Schewe and B. Thalheim, editors, *International Workshop on Semantics in Data and Knowledge Bases, SDKB 2008, Nantes, France, March 29, 2008, Proceedings*, volume 4925 of *Lecture Notes in Computer Science*, pages 172–191. Springer-Verlag, 2008.
19. A. Hutchinson. Metrics on terms and clauses. In M. van Someren and G. Widmer, editors, *Machine Learning: ECML-97, 9th European Conference on Machine Learning, Prague, Czech Republic, April 23-25, 1997, Proceedings*, volume 1224 of *Lecture Notes in Computer Science*, pages 138–145, 1997.
20. A. M. Keller. *Updating Relational Databases through Views.* PhD thesis, Stanford University, 1985.
21. R. Langerak. View updates in relational databases with an independent scheme. *ACM Trans. Database Systems*, 15(1):40–66, 1990.
22. J. Lechtenbörger. The impact of the constant complement approach towards view updating. In *Proceedings of the Twenty-Second ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, San Diego, California, June 09-11, 2003*, pages 49–55, 2003.
23. D. Maier. *The Theory of Relational Databases.* Computer Science Press, 1983.
24. S.-H. Nienhuys-Cheng. Distance between Herbrand interpretations: A measure for approximations to a target concept. In *Inductive Logic Programming, 7th International Workshop, ILP-97, Prague, Czech Republic, September 17-20, 1997, Proceedings*, volume 1297 of *Lecture Notes in Computer Science*, pages 213–226. Springer, 1997.

Data Dependencies for Access Control Policies

Romuald Thion[1] and Stéphane Coulondre[2]

[1] INRIA Grenoble – Rhône-Alpes, France
655 Av. de l'Europe, Montbonnot
38334 Saint Ismier Cedex, France
[2] University of Lyon, LIRIS, France
Bât. Blaise Pascal (501), 20, Av. A. Einstein
F-69621 Villeurbanne, France

Abstract. Access control policies are set of facts and rules that determine whether an access request should be granted or denied. Policies must satisfy a set of constraints to reflect some high level organization requirements. First-order logic has been advocated for some time as a suitable formal framework for access control policies. However, though formally expressed, constraints are not defined in a unified language that could lead to some well-founded and generic enforcement procedures.
Therefore, we directly start by proposing to express access control constraints in an unified and generic way by mean of data dependencies. We show how to use well-founded procedures dedicated to dependencies (*chases*) to enforce and reason on constrainted policies. Without requiring any rewriting previous to the inference process, our approach provide clean and intuitive debugging traces for security officers and is generic enough to capture expressive access control policies.

1 Introduction

Security policies are sets of laws and rules governing the security of organizations. Access Control (AC) (or authorization) policies are specialized forms of security policies, dedicated to logical right management. Since the RBAC initiative [1], several models have been proposed to organize rights in an intuitive manner. Some capture contextual conditions (e.g. *Generalized-Temporal*-RBAC [2] or *Geographical*-RBAC [3]), others have introduced new concepts to organize right and to stick with organization (e.g. Workflow-RBAC [4], *Organization*-BAC [5]). Throughout these propositions, fragments of First-Order Logic (FOL) has been advocated as a general framework suitable to formalize AC.

In addition to innovative concepts and relations to organize rights, AC models have integrated *constraints* to reflect some high level organization requirements that must be enforced within policies. The most prominent one is the *mutual exclusion*. For instance, the definition of roles *clerk* and *manager* as mutually exclusive in an RBAC policy means that *no user* should be assigned to both *clerk* and *manager*.

With the development of AC models, several kinds of constraints have been defined: variations of mutual exclusion, prerequisite constraints or constraints

over hierarchies [6]. All of these different classes of constraints share a common objective: to restrict the set of policies expressible over a model to the set of *consistent ones*. Unfortunately, constraints are thought for specific models and not considered as first class citizen of AC in a generic way.

In order to express contraints in an homogeneous way, a formal language able to deal with broad classes of constraints is necessary. Such a language should be integrated in a model that is flexible enough to express general AC policies. It should allow the definition of *new classes* of constraints and should be able to capture general *integrity requirements* of AC models. This language must have clear semantics, and provide well-founded automated proof procedures for *checking* consistency of policies, in order to ensure global consistency.

To address these issues, we propose a generic logical framework for constraints. AC policies are formally defined in a general logical setting grounded on database theory (section 2). The formal framework is able to cope with common models and extensions found in the literature which are formalized in fragments of FOL (e.g. DATALOG, Horn clauses). We make use of *relational data dependencies* to model AC constraints. Dependencies are able to capture complex integrity requirements and constraints of AC policies in an homogeneous way (section 3).

We define a set of well-founded operations that can be used to help AC models designers and policy administrators in making constraints design and administration easier. These operation are based upon *chases* for data dependencies (section 4). We have implemented the framework and validated our approach with automated formal proofs of some previous results from the literature that had been manually proved. Moreover, the proofs obtained are quite readable as no prior rewriting is performed (such as clausal form), for it may obfuscate human analysis (section 5).

2 Access control models

First of all, without lack of generality, we operate a distinction between *models* and *policies*: AC models defines structures and AC policies are instances of these structures. Policies are *logical models* (in the model-theoretic sense) of a theory defined by an AC *model*. The word *model* is indeed prone to confusion. We use the term (AC) *model* to refer to the structure that describes how rights are organized and granted (i.e. the meaning of *model* in the AC literature). We use the term *logical model* to refer to a model-theoretic interpretation which satisfies a set of closed FOL formulae.

2.1 Access control background

An AC model relies on *vocabulary*: a set of *sorts* and *relations* between sorts. Sorts are the main concepts used to organize rights (e.g. users, roles, resource ...). Relations define how sorts are related (e.g. assignments between users and roles). A *subject* is the sort that represents a (physical) user in the system and that acts on his behalf.

Sorts partition the set of constants used in a policy. This property is tied to the *many-sorted* FOL framework. It is shown that many-sorted FOL can be reduced to one-sorted FOL (i.e. classical FOL) by assigning a specific unary predicate symbol D_S called *domain predicate symbols* to each sort S [7, chapter 10, p. 460]. In our framework, we implicitly operate this transformation by assigning a unique unary predicate symbol to each sort.

Definition 1. *Access control vocabulary. The* vocabulary *of an* AC *model is the union of a set Sorts of* unary predicate symbols *called* sorts *and a set Rel of* n-ary predicate symbols *called* relations.

The sorts of users *(User),* subjects *(Subject),* actions *(Action) and* objects *(Object) must be defined in any* AC *vocabulary.*

The vocabulary is partitioned into two sets edb, called core vocabulary*, and idb, its* intensive vocabulary*:* $idb \cap edb = \emptyset$ *and* $idb \cup edb = Voc$.

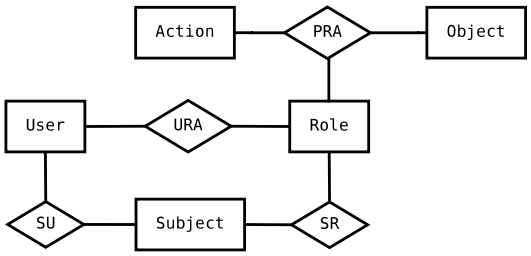


Fig. 1. The core vocabulary *edb* of RBAC.

Figure 1 illustrates the core vocabulary of RBAC models. This vocabulary is composed of five sorts (drawn by rectangles in figure 1): $User$, $Subject$, $Role$, $Action$ and $Object$. Moreover, four relations are defined (drawn by diamonds in figure 1): URA betwen $User$ and $Role$, PRA between $Role$, $Action$ and $Object$, SU between $Subject$ and $User$ and SR between $Subject$ and $Role$.

The *state* of an AC model is a set of facts defined over the core vocabulary *edb*. The state can practically be stored in a Relational Database Management System (RDBMS). Following traditional axioms of logical interpretation of relational databases, we assume that constants are distinct and that states are finite. In the context of RBAC, the term AC *state* (a.k.a. RBAC *database*) has first been coined in [8].

Definition 2. *Access control state. To each sort* $S \in edb$ *is associated a set of constants* $\mathbf{S}$ *called its* domain*. Domains are pairwise disjoint. To each relation* $R \in edb$ *of arity* n *between sorts* $S_1 \ldots S_n$*, is associated the set* $\mathbf{R} = \mathbf{S}_1 \times \ldots \times \mathbf{S}_n$.

An access control state $\mathbf{I}$ *on an* AC *vocabulary* Voc *is a mapping from each sort* $S \in edb$ *to a subset of* $\mathbf{S}$*, called its* active domain*, and from each relation* $R \in edb$ *to a subset of* $\mathbf{R}$.

For instance, a state over *edb* shown in figure 1 contains user-role, permission-role, session-user and session-role assignements, e.g. respectively $URA(\texttt{Bob}, \texttt{r}_1)$, $PRA(\texttt{r}_1, \texttt{r}, \texttt{file2})$, $SU(\texttt{S2}, \texttt{Bob})$ and $SR(\texttt{S2}, \texttt{r}_1)$.

Rules expressed over the vovabulary describes how effective rights (`Bob` is granted `r` on `file2` because he endorses the role $\texttt{r}_1$) are derived from the state. Several fragments of FOL have been used as formal languages to capture AC rules. DATALOG-based models are considered expressive enough to capture complex AC policies [5, 9–12], thus, we restrict ourselves to DATALOG.

Definition 3. *Access control rules and policy. The* rules *of an* AC *model is a set P of* DATALOG *sentences.*

Given an AC *state* $\mathbf{I}$ *of a model* $AC = (Voc, P)$*, an* access control policy $\mathbf{I}'$ *over a given state* $\mathbf{I}$ *is the minimal logical model of* P*:* $\mathbf{I}' \models P$ *with* $\mathbf{I} \subseteq \mathbf{I}'$*. Computationally,* $\mathbf{I}'$ *is a fixpoint of* P *considered as a (*DATALOG*) program [13, theorem 12.5.2, p. 301].*

2.2 Main relations of access control

An AC request is a triple subject, action, and object: $(s, a, o) \in Subject \times Action \times Object$. A reference monitor enforces AC policy, it takes a boolean decision upon an AC request. In our setting we deal only with positive permissions and supposes the closed world hypothesis: any access which is not explicitly granted is denied.

Definition 4. *Decision triples.* $Access \subseteq Subject \times Action \times Object$ *is set of* AC *permissions granted to subjects.* $Access$ *is computed from* P *and* $\mathbf{I}$*. An access control request* $(s, a, o) \in Subject \times Action \times Object$ *is granted iff* $(s, a, o) \in Access$*.*

Hierarchies been introduced in AC to reduce the number of assignments in policy. For instance, in RBAC roles are given a preorder (reflexive, transitive) $\preccurlyeq$ modelling an *is a* relationship. Relation $r_1 \preccurlyeq r_2$ means that every permissions granted to role r_1 are granted to r_2 and that each user who is member of role r_2 is also member of r_1. For example, the hierarchy $\texttt{employee} \preccurlyeq \texttt{accountant} \preccurlyeq \texttt{manager}$ may be defined in a financial organization.

Definition 5. *Inheritance. A sort* $S \in edb$ *of an* AC *model is given an* inheritance *relationship* $Senior_S \subseteq S \times S$ *if a* dominance *relation (its skeleton)* $SeniorD_S \subseteq S \times S$ *is defined in edb and if* $Senior_S$ *is defined by means of* P *as the reflexive and transitive closure of* $SeniorD_S$*.*

Another important feature introduced in the AC literature is *mutual exclusion*. Mutual exclusion is a technical mean to enforce separation of duties. As it is the case for hierarchies, mutual exclusion needs two relations: an extensive one and an intensive one which is its symmetric closure. Moreover, when both an exclusion and an inheritance relation have been defined on the same sort, an additional rule must be settled P, stating that exclusion is propagated via inheritance [8].

Definition 6. *Mutual exclusion. A sort $S \in edb$ is given a* mutual exclusion *relationship $Mutual_C \subseteq S \times S$ if a core separation relation $MutualD_C \subseteq S \times S$ is defined in edb and if $Mutual_C$ is defined by means of P as the symmetric closure of $SeniorD_S$.*

If an inheritance relation $Senior_S$ is also set on sort S, then the following rule must be defined:

$$Mutual_S(S, S') \wedge Senior_S(S'', S) \Rightarrow Mutual_S(S, S'')$$

3 Constraints in access control policies

Among the formal tools developed by the database community, *data dependencies* (a.k.a *integrity constraints*) have been defined to capture integrity contraints over relational data. In a unification attempt, they have been defined as FOL sentences [13, Chapter 10]. The best known classes are *functional* (FD), *inclusion* (IND) and *multivalued* (MVD) dependencies.

Constrained Tuple-Generating (CTGD) [14] and *Disjunctive*-CTGD (DCTGD) dependencies [15] are among more recent classes introduced. These classes have been developed to express complex statements on relational data: they can model semantic relationships in spatial, temporal or multimedia databases. We benefits their increased expressivity by modelling complex and subtle AC constraints.

In one of their general form (CTGD), dependencies are FOL sentences having the following syntax (we use standard conventions of logic applied to relational databases, where R_i and Q_i are relation symbols, ψ and ϕ are conjunction of constraints):

$$\forall \tilde{X}\ R_1(X_1) \wedge \ldots \wedge R_n(X_n) \wedge \psi(\tilde{X}) \Rightarrow$$
$$\exists \tilde{Z}\ Q_1(Y_1) \wedge \ldots \wedge Q_m(Y_m) \wedge \phi(\tilde{Y})$$

where $\tilde{Z}$ does not designate the whole set of variables in the head but only those which are not already bound by a universal quantifier ($\tilde{Z} = \tilde{Y} - \tilde{X}$). We use $\perp$ as a logical antilogy (e.g. $0 = 1$).

Dependencies can be used either to restrict authorized values in a policy (e.g. FD), or to impose presence of tuples if some other ones are already present in the policy (e.g. IND, MVD): they express conditions that *must hold* in the policy.

In our framework, the set of constraints over an AC model is denoted as Σ. This set is generic enough to capture desirable properties which cannot be expressed in P. From the logical perspective $P \cup \Sigma$ is a logical theory, i.e. a set of closed FOL formulae. Depending on the expressivity of Σ, increasingly complex types of AC constraints can be captured.

Definition 7. *Access control model. An access control model is a triple $AC = (Voc, P, \Sigma)$, where Σ is a set of constraints expressed as* data dependencies.

The semantic of constraints is given by the standard FOL *model-theoretic interpretation of dependencies. An access control policy $\mathbf{I}'$ built from a state $\mathbf{I}$ is* consistent *iff $\mathbf{I}' \models \Sigma$.*

(1)	$SR(S,R) \Rightarrow \exists U\ SU(S,U)$
(2)	$SU(S,U) \wedge SU(S,U') \Rightarrow U = U'$
(3)	$SU(S,U) \wedge SR(S,R) \Rightarrow URA(U,R)$
(4)	$SR(U,R) \wedge SR(U,R') \Rightarrow R = R'$

Table 1. Integrity statements for subject sort in RBAC.

The next subsections describe how dependencies are used to express various security requirements of models in an homogeneous way: integrity of states, algebraic properties that cannot be captured by rules, semantic of mutual exclusion, constraints on authorizations and administrative prerequisites.

3.1 Integrity constraints on states

Integrity constraints on states ensure that the extensive part of an AC policy is consistent according to a set of basic constraints expressed over *edb*. It leads to the definition of *well-founded state*. Theses constraints ensure generalized primary key and foreign key constraints on core concepts of AC models.

Definition 8. *Well-founded state. Let Λ be a set of constraints $\Lambda \subseteq \Sigma$ involving* only *symbols of edb. Let* $\mathbf{I}$ *be a state. Then* $\mathbf{I}$ *is* well-founded *iff* $\mathbf{I} \models \Lambda$.

For instance, in the RBAC standard it is defined that each subject have to be assigned to a *unique* user (expressions (1) for existence and (2) for uniqueness in table 1) and that a role can be used by a subject *only if* the role is assigned to the user who owns the subject (expr. (3)). Moreover, it has been argued that "The (RBAC) standard should accommodate RBAC systems that allow only one role to be activated in a session" [16] (expr. (4)).

3.2 Algebraic constraints of relations

AC models hierarchies are commonly defined as partial orders to avoid cycles. Moreover, a mutual exclusion is defined as irreflexive in order to prevent a sort from being mutually exclusive with itself. Antisymmetry and irreflexivity can be expressed by dependencies by Constraint-Generating Dependencies (CGD) (expressions (1) and (2) of table 2).

Moreover, some additional constraints can be set over an inheritance relationship. Some models specialize hierachies over sorts to be trees (expr. (3)), inverted trees (expr. (4)) or lattices (expr. (5) and (6)) using algebraic properties of $Senior_S$ and $SeniorD_S$ relations.

3.3 Semantics of mutual exclusion

Dependencies can capture the semantics of mutual exclusion. Nullity-Generating Dependencies (NGD) (a.k.a implication constraints) are of special interest for this

(1)	$Senior_R(R, R') \wedge Senior_R(R', R) \Rightarrow R = R'$
(2)	$Mutual_R(R, R) \Rightarrow \bot$
(3)	$SeniorD_S(S, S'), SeniorD_S(S, S'') \Rightarrow S' = S''$
(4)	$SeniorD_S(S', S), SeniorD_C(S'', S) \Rightarrow S' = S''$
(5)	$Sort(S'), Sort(S'') \Rightarrow \exists S_\bot\ Senior_S(S', S_\bot), Senior_S(S'', S_\bot)$
(6)	$Sort(S'), Sort(S'') \Rightarrow \exists S_\top\ Senior_S(S_\top, S'), Senior_S(S_\top, S'')$

Table 2. Common algebraic properties of relations.

purpose. Several interpretations of mutual exclusion exist. For instance, in RBAC policy, to set that r and r' are mutually exclusive may have up to four different meaning [6]:

- no *user* could be both assigned roles r and r' (expression (1) of table 3),
- no *subject* could be both assigned r and r' (expr. (2)),
- no common *permission* could be granted to both r and r' (expr. (3)),
- no action on a common *object* could be granted to both r and r' (expr. (4)).

3.4 Constraints on decision triples

Tuples-generating dependencies can express restrictions on decision triples. The RBAC model imposes that any authorization must be granted via a role [1]:

> **Property 3.2**: *"A subject s can perform an operation op on object o* only if *there exists a role r that is included in the subject's active role set and there exists an permission that is assigned to r such that the permission authorizes the performance of op on o".*

Defining intermediate sorts between users and permissions is a way to simplify administration tasks. Bypassing these sorts is error-prone and may lead to ambiguities within policies. To the best of our knowledge, this property is neither captured in logic-based modelling attempts of RBAC, nor in extended models.

Definition 9. *Property of decision triples. If there is a rule in P of the form $\psi \Rightarrow Access$, then there is a corresponding dependency in Σ of the form $Access \Rightarrow \psi$ that makes the rule an* if and only if *condition.*

(1)	$Mutual_R(R, R') \wedge URA(U, R) \wedge URA(U, R') \Rightarrow \bot$
(2)	$Mutual_R(R, R') \wedge SR(S, R) \wedge SR(S, R') \Rightarrow \bot$
(3)	$Mutual_R(R, R) \wedge PRA(R, A, O) \wedge PRA(R', A, O) \Rightarrow \bot$
(4)	$Mutual_R(R, R') \wedge PRA(R, A, O) \wedge PRA(R', A', O) \Rightarrow \bot$

Table 3. Constraints related to mutual exclusion in RBAC.

For instance, if the rule $SR(S,R) \wedge PRA(R,A,O) \Rightarrow Access(S,A,O)$ defines $Access$, then transposition of property (3.2) is $Access(S,A,O) \Rightarrow \exists R\, SR(S,R) \wedge PRA(R,A,O)$.

Note that these constraints do not prevent an administrator from defining discrete authorizations. However, they enforce that only *redundant authorizations*, which can be derived from $\mathbf{I}$ and P, can be set.

3.5 Administrative prerequisite

An administrative prerequisite enforce the presence of tuples *before allowing administrative operations* [1]. Administrative operations consist in updates, insertions and deletions of tuples within the state $\mathbf{I}$. If any administrative prerequisite, is violated, the transaction initiated by the administrator will not be committed. We model prerequisite constraints, with a relation $Required_{Sort} \subseteq Sort \times Sort$. Informally, $Required_{Sort}(\mathtt{a},\mathtt{b})$ means that a an b cannot exists without a a.

Definition 10. *Administrative prerequisite. An administrative prerequisite constraint imposes the presence of tuples in* $\mathbf{I}$ *or* $\mathbf{I}'$*. A transitive prerequisite relation* $Required_R \in idb$ *over a relation* R *is defined as the transitive closure of a relation* $RequiredD_R \in edb$*.*

Prerequisite constraints are modelled as (constrained) TGD *in* Σ *by an auxiliary relation* $RequiredD_R$ *of the following form:*

$$\forall \tilde{X}\ R(X_R) \wedge \ldots \wedge Required_R(X) \wedge \phi(\tilde{X}) \Rightarrow \exists \tilde{Z}\ R(X'_R) \wedge \ldots \wedge \phi(\tilde{Y})$$

For instance, in the RBAC models, administrative prerequisites are used to prevent administrators from assigning roles to users if some role has not already been defined (e.g. no *assistant manager* without a *manager*). This is an example of a prerequisite over roles. The following sentences express these statements by means of the URA relation. Defining $RequiredD(\mathtt{assistant},\mathtt{manager})$ in $\mathbf{I}$ imposes that whenever a user is an assistant manager, then there must be at least one user who is a manager.

$$RequiredD_{Role}(R,R'), Required_{Role}(R',R'') \Rightarrow Required_{Role}(R,R'')$$
$$URA(U,R), Required_{Role}(R,R') \Rightarrow \exists U'\ URA(U',R')$$

4 Constraint verification

The task of defining Voc, P and Σ is dedicated to the AC model designer. Security officers define the state $\mathbf{I}$, and $\mathbf{I}' = P(\mathbf{I})$ is computed from $\mathbf{I}$ and P. The policy is consistent if and only if $\mathbf{I}' \models \Sigma$. Thus, from an abstract perspective, there are only a few differences between P and Σ which can be considered as a whole as a logical theory $T = P \cup \Sigma$.

As it is case in the relational paradigm, the main difference between P and Σ lays in their usage: P is used to *compute* the policy from the state, whereas Σ is used to impose *restrictions* on authorized instances of $\mathbf{I}'$.

In this section we rely on two theoretical problems over logical theories:

– the *satisfaction problem*. Answering whether a policy $\mathbf{I}'$ satisfies a given FOL sentence σ: $\mathbf{I}' \models \sigma$. This problem is central for computing $\mathbf{I}'$ from $\mathbf{I}$, for answering queries and for checking whether a policy satisfies the set of constraints for a given model.
– the *logical implication problem*. Answering whether a set of closed formulae T logically implies a closed formula σ: $T \models \sigma$ or, in other words, deciding if *any* policy is also a model of the single sentence σ. This problem is central for simplifying logical theory $P \cup \Sigma$ or for checking model consistency from an abstract perspective, *without considering any state*.

We have implemented the proof procedures for several rich classes of dependencies to validate our approach. As shown in section 5, our prototype allows automatizing the consistency checking of RBAC policies and furnishing proof of previous results proposed in the AC literature [8].

4.1 Satisfaction problem

One of the features expected from an access control model implementation is the *administrative review*. In the present framework, administrative reviews are simple conjunctive queries over $\mathbf{I}'$. Therefore the requirements of the standard definition are met [1, 11]. Example of an adminsitrative review is the query $\{(u, a, o) \mid \exists r\, URA(u, r) \wedge PRA(r, a, o)\}$, returning the set of permissions granted to users through their roles.

Definition 11. *Administrative reviews. An* administrative review *is any conjunctive query (from the standard database sense) built upon* Voc.

The validity of an AC policy is checked by verifying, given a model $AC = (Voc, P, \Sigma)$, whether $\mathbf{I}' \models \Sigma$. Unsatisfaction of a set of constraints by a policy can fall into two categories:

– the policy is *inconsistent*: some constraints-generating dependencies are not satisfied. For instance, a policy is said to be inconsistent if antisymmetry, irreflexivity or exclusion relations are not satisfied,
– the instance is *incomplete*: some tuples-generating dependencies are not satisfied. For instance, a policy is said to be incomplete if some properties of authorization relations or administrative prerequisites are not satisfied.

If a policy is *inconsistent* or *incomplete*, administrators have to correct the state. Whenever a policy is inconsistent, deletion of existing facts or value updates should be favoured. Whenever a policy is incomplete, addition of facts should be privileged.

4.2 Logical implication problem

Whereas previous subsection have been devoted to policy checking, this section considers AC models from an abstract perspective, without reference to *any* particular state. The main problem we address is simplification of AC models. In

the case of a new tailored AC model, where many collaborative designers from different sites might be involved, the associated logical theory may become quite large. Thus, for practical purposes, it is necessary to reduce the size of the theory.

Definition 12. *Redundancy in an* AC *model. Let be* $T = P \cup \Sigma$ *the logical theory of an* AC *model. Let be* $\sigma \in T$*, if* $T \backslash \{\sigma\} \models \sigma$ *then the dependency* σ *is said* redundant*, moreover* $T \backslash \{\sigma\}$ *and* T *have the same models.*

The authors of [14] give two bottom-up chase procedures for solving the implication problem of CTGD: given a set of CTGD Σ, and a single CTGD σ (of the form $l \Rightarrow r$), determine whether in every policy where Σ is satisfied, σ is also satisfied, stated briefly as $\Sigma \models \sigma$. The operational nature of proof procedures for CTGD is based on the concept of tuple (a grounded atom, with no variables). They keep the same strategy as the original chase of [17] extended to deal with constraints.

The CTGD implication problem is semi-decidable: the procedures may run forever. However, there are some interesting decidability results holding in various subclasses of CTGD. For example, the chase is decidable for TGD having no existentially quantified variable [17]. For implementation purposes though, bounding up the number of closure operator applications we be wanted, but this is still uncertain.

5 Application

σ_1 $URA(User, Role_1), URA(User, Role_2), Mutual(Role_1, Role_2) \Rightarrow \bot$
σ_2 $Mutual(Role, Role) \Rightarrow \bot$
σ_3 $Mutual(Role_1, Role_2) \Rightarrow Mutual(Role_2, Role_1)$
σ_4 $Senior(Role_1, Role_2), Mutual(Role_1, Role_2) \Rightarrow \bot$
σ_5 $Mutual(Role_1, Role_2), Senior(Senior, Role_1), Senior(Senior, Role_2) \Rightarrow \bot$
σ_6 $Senior(Senior, Role_1), Mutual(Role_1, Role_2) \Rightarrow Mutual(Senior, Role_2)$.

Table 4. Logical characterization of RBAC constraints, adapted from [8]

We have written a prototype C++ toolkit library (LIBDEPENDENCIES) for inference on dependencies. It can handles the CTGD as well as its subclasses. Four different *chases* have been implemented [14, 17, 18] in the toolkit.

The authors of [8] have defined a set of integrity properties for RBAC models. These integrity requirements are captured by dependencies in table 4. The authors have manually proved that the set of properties of the table can be reduced to a smaller set.

Let examplify a run of the LIBDEPENDENCIES that uncover results from [8]. Initially, the LIBDEPENDENCIES is loaded with dependencies σ_1 through and σ_6 of table 4. The goal is to prove that $\{\sigma_2, \sigma_3, \sigma_6\} \models \sigma_5$: the following trace is a formal proof of this entailment which is obtained using LIBDEPENDENCIES:

1. let be $Mutual(\mathtt{r}_1, \mathtt{r}_2)$, $Senior(\mathtt{r}_0, \mathtt{r}_1)$ and $Senior(\mathtt{r}_0, \mathtt{r}_2)$ tuples,
2. by σ_3, derive $Mutual(\mathtt{r}_2, \mathtt{r}_1)$,
3. by σ_6 twice, derive $Mutual(\mathtt{r}_0, \mathtt{r}_2)$ and $Mutual(\mathtt{r}_0, \mathtt{r}_1)$,
4. by σ_3 twice, derive $Mutual(\mathtt{r}_2, \mathtt{r}_0)$ and $Mutual(\mathtt{r}_1, \mathtt{r}_0)$
5. by σ_6, derive $Mutual(\mathtt{r}_0, \mathtt{r}_0)$ (there are two ways),
6. finally, by σ_2 derive $\bot$.

Thus, the chase procedure proved that $\{\sigma_2, \sigma_3, \sigma_6\} \models \sigma_5$. The prototype LibDependencies can be used to derive other sample theorem from the dependencies of table 4. Let be Σ the six dependencies show in this table. The chase procedures can prove that $\Sigma \backslash \{\sigma_4\} \models \sigma_4$, $\Sigma \backslash \{\sigma_5\} \models \sigma_5$ or even that $\Sigma \backslash \{\sigma_4, \sigma_5\} \models \sigma_4, \sigma_5$. Such a result is AC model independent and can be applied for any right management system built as $AC = (Voc, P, \Sigma)$.

This example illustrates the utility of our approach. Given an AC model $AC = (Voc, P, \Sigma)$, automated proof of non-trivial properties can be provided. Besides the above obtained theoremsfrom [8], we have been able to derive the following results:

- *read and write* access over an objet in Mandatory Access Control (Mac) are granted to a subject iff the subject's clearance level is equal to the object's confidentiality level,
- a *root role* which inherits all other ones cannot exist in an Rbac policy where two roles are mutually exlusive from [19],
- dynamic authorizations are a subset of static authorizations in Rbac policies from [1].

6 Related work

The authors of [10, 12] have used respectively C-Datalog (which introduce object-oriented concepts), and Datalog^C. The basic components of this paper reuse and extend some their formal models of AC. The authors of [9, 20, 21] have extended Datalog with specific constructs which are reduced into standard Datalog formulae. However, we have not used rewriting procedures which may obfuscate the debugging steps thus puzzle administrators.

The authors of [11, 22] describe Fol programs to deal with AC. They address the problems arising of *hybrid* policies in which both authorizations and denial can defined.

These frameworks allow a permissive use of negation in formal sentences, whereas we have chosen to favour existential quantifiers. This choice enables modeling of complex integrity requirements considered as fundamental in Rbac, which are not expressible in other frameworks.

Constraints have received lots of attention in AC models. However, most of attention is dedicated to *separation of duties* and related constraints [8, 6]. The algebra of [23] consider separation of duties constraints as high-level organization requirements. Our approach is quite different: constraints first-class citizien

Symbol	Database	Access control
$AC = (Voc, P, \Sigma)$	*deductive database*	*access control model*
P	DATALOG *rules*	principles of model
Σ	data dependencies	constraints
$\mathbf{I}$	extensive database	state
$\mathbf{I}'$	intensive database	policy

Table 5. Relations between databases and access control.

of AC models. Thus we can capture intrinsic properties (e.g. constraints on decision triples, prerequisites) as well as general integrity requirements (e.g. states well-foundedness, algebraic properties of relations) which are usually taken into account separately.

Our main inovation is the fruitful use of data dependencies as a unifying logical framework which encompasses both traditional AC rules *and* integrity constraints. Thus, constraints are expressed in the very same model, and not expressed in an independent and different framework. As constraints are integrity requirements of policies, we argue that their integration as soon and as tightly as possible in the model is a step towards ensuring AC *robustness*. Moreover, our framework is clearly grounded on foundation of databases and makes a clear distinction between policies and models. This separation, advocated in [16], is not explicit in related work.

We rely on some known results for data dependencies, in order to provide well-founded tools for reasoning on policies. An interesting feature of these proof procedures is that they do not require any prior translation of FOL formulae of P and Σ (e.g. by means of Skolemization or rewriting rules). This property leads to native clear traces of automated proofs, as given in section 5. This greatly enhances the readability of inference results, for design and maintenance purposes. Finally, by mean of the LIBDEPENDENCIES, we have been able to re-prove in an automated way some interesting results found in the literature.

7 Discussion and conclusion

We have presented a framework for AC policies which relies on logical aspects of databases. The key idea is not to deploy AC policies *in* databases, but to express and handle AC policies *by mean* of database theory. Our approach provides an homogeneous way for defining both AC rules and integrity requirements within the very same logic background. The relations between logic in databases and AC are summarized by table 5.

The logical fragment of FOL we used to define P and Σ is quite restrictive: negation, disjunction and function symbols are not allowed. The main argument is that we obtain a *unique* and *computable* model of P. Computability is necessary because the reference monitor has to answer each access request.

Furthermore, we chose to have a more expressive framework for *constraints* than for *rules*, by favouring *existential quantification* over *negation*. The main goal is to be able to model some of the most important constraints commonly identified for AC models, which cannot be expressed in DATALOG models. Even with a quite simple AC model (for instance, with a few sorts, relations and principles), most of the properties given in section 3 require existentially quantified variables (e.g. sessions integrity properties, restricted hierarchies, authorizations properties and administrative prerequisites). We are currently investigating extensions of our proposal:

- extension of the language of P and Σ, to capture new model properties (e.g. spatio-temporal based authorization usually needs arithmetic constraints in hypothesis of rules). However, algorithms decidablity and tractability should be taken into account. As an example, it may be interesting to use some decidable subclass of TGD by imposing some restrictions on existential quantification,
- exploring databases automated maintenance [24], and data integration [25] to fix non-consistent policies. Data integration may be a fruitful perspective for composition of policies expressed in different models [26],
- another emerging topic is *usage control* and *privacy protection*. The basic components and definitions we presented can be used to define next generation AC models and policies, as it has been done with RBAC models [27].

Acknowledgement : this work was supported by Région Rhône-Alpes.

References

1. Ferraiolo, D.F., Kuhn, R.D., Chandramouli, R.: Role-Based Access Control. Artech House Publishers (2003)
2. Joshi, J., Bertino, E., Latif, U., Ghafoor, A.: A generalized temporal role-based access control model. IEEE Transactions on Knowledge & Data Engineering **17**(1) (2005) 4–23
3. Damiani, M.L., Bertino, E., Catania, B., Perlasca, P.: GEO-RBAC: A spatially aware rbac. ACM Transactions on Information & System Security **10**(1) (2007)
4. Wainer, J., Kumar, A., Barthelmess, P.: DW-RBAC: A formal security model of delegation and revocation in workflow systems. Information Systems **32**(3) (2007) 365–384
5. Miège, A.: Dénition d'un environnement formel d'expression de politiques de sécurité : modèle Or-BAC et extensions. PhD thesis, Ecole Nationale Supérieure des Télécommunications,Paris (2005)
6. Crampton, J.: Specifying and enforcing constraints in role-based access control. In: SACMAT'03: 8th ACM Symposium on Access Control Models and Technologies, ACM Press (2003) 43–50
7. Gallier, J.H.: Logic for Computer Science: Foundations of Automatic Theorem Proving. Revised on-line version 2003. Harper & Row (1986) http://www.cis.upenn.edu/ jean/gbooks/logic.html.

8. Gavrila, S.I., Barkley, J.F.: Formal specification for role based access control user/role and role/role relationship management. In: RBAC'98: 3rd ACM workshop on Role-based access control. (1998) 81–90
9. Jim, T.: SD3: A trust management system with certified evaluation. In: IEEE Symposium on Security and Privacy. (2001) 106–115
10. Bertino, E., Catania, B., Ferrari, E., Perlasca, P.: A logical framework for reasoning about access control models. ACM Transactions on Information & System Security **6**(1) (2003) 71–127
11. Barker, S., Stuckey, P.J.: Flexible access control policy specification with constraint logic programming. ACM Transactions on Information & System Security **6**(4) (2003) 501–546
12. Li, N., Mitchell, J.C.: DATALOG with constraints: A foundation for trust management languages. In Dahl, V., Wadler, P., eds.: PADL'03: 5th International Symposium on Practical Aspects of Declarative Languages, New Orleans. Volume 2562 of Lecture Notes in Computer Science., Springer-Verlag (2003) 58–73
13. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley, Boston (1995)
14. Maher, M.J., Srivastava, D.: Chasing constrained tuple-generating dependencies. In Hull, R., ed.: PODS'96: 15th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Montreal, Canada, ACM Press (1996) 128–138
15. Wang, J.: Exploiting Constraints for Query Processing. PhD thesis, Griffith University, Brisbane, Queensland, Australia (2002)
16. Li, N., Byun, J.W., Bertino, E.: A critique of the ANSI standard on role-based access control. IEEE Security and Privacy **5**(6) (2007) 41–49
17. Beeri, C., Vardi, M.Y.: A proof procedure for data dependencies. Journal of the ACM **31**(4) (1984) 718–741
18. Coulondre, S.: A top-down proof procedure for generalized data dependencies. Acta Informatica **39**(1) (2003) 1–29
19. Benantar, M., ed.: Access Control Systems - Security, Identity Management and Trust Models. Springer-Verlag (2006)
20. Li, N., Grosof, B.N., Feigenbaum, J.: Delegation logic: A logic-based approach to distributed authorization. ACM Transactions on Information & System Security **6**(1) (2003) 128–171
21. DeTreville, J.: Binder, a logic-based security language. In: SP'02: IEEE Symposium on Security and Privacy, Washington, IEEE Computer Society (2002) 105
22. Halpern, J.Y., Weissman, V.: Using first-order logic to reason about policies. In: CSFW'03: 16th IEEE Computer Security Foundations Workshop, Pacific Grove, California, IEEE Computer Society (2003) 187–201
23. Li, N., Wang, Q.: Beyond separation of duty: An algebra for specifying high-level security policies. J. ACM **55**(3) (2008)
24. Chomicki, J., Marcinkowski, J.: Minimal-change integrity maintenance using tuple deletions. Information & Computation **197**(1-2) (2005) 90–121
25. Fagin, R.: Inverting schema mappings. In Vansummeren, S., ed.: PODS'06: 25th ACM SIGACT-SIGMOD-SIGART Symposiumon Principles of Database Systems, Chicago, Illinois, ACM Press (2006) 50–59
26. Li, N., Wang, Q., Qardaji, W., Bertino, E., Rao, P., Lobo, J., Lin, D.: Access control policy combining: theory meets practice. In: SACMAT '09: Proceedings of the 14th ACM symposium on Access control models and technologies, ACM (2009) 135–144
27. Ni, Q., Bertino, E., Lobo, J.: Privacy-aware RBAC - leveraging RBAC for privacy. IEEE Security and Privacy **to appear** (2009)

Comparing Availability in Controlled Query Evaluation Using Unordered Query Evaluation for Known Potential Secrets

George Voutsadakis[1,2]

[1] Department of Computer Science, Iowa State University, Ames, IA, USA
[2] Department of Mathematics and Computer Science, Lake Superior State University, Sault Ste. Marie, MI, USA

Abstract. Controlled Query Evaluation (CQE) is an inference control mechanism used to dynamically preserve confidentiality in secure information systems. In this note we introduce the notion of unordered query evaluation as a vehicle for comparing availability of various CQE methods, a recurring theme in the CQE literature. Moreover, we show that the various procedural approaches introduced thus far in the literature for imposing the declarative requirements of CQE lead to maximally available mechanisms. Finally, we characterize maximally available unordered query evaluation for various enforcement methods for known potential secrets as the unordered query evaluation resulting from CQE mechanisms for some suitably chosen ordering of the queries.

1 Introduction

Preserving confidentiality in information systems that contain both classified and public data is one of the major goals in data security. Two general categories of methods have been considered in the literature: *access control* and *information flow control*, that are characterized by their *static* and *dynamic* nature, respectively. In using access control, one of the major challenges is the *inference problem*, i.e., the potential inference of secret information by the user, based on revealed public information. Some pointers to the literature on the inference problem in access control are [11, 13, 19] (see [12] for a review). This problem is addressed in *Controlled Query Evaluation* (CQE), an inference control mechanism used to dynamically preserve confidentiality [14]. In CQE, the knowledge base administrator specifies the information that is to be kept confidential. When a query is posed against the knowledge base, its true answer is computed and, before an answer is issued to the user, a *censor* is used, aided by a *log* maintained to represent the user's assumed information, to detect potential security risks. If the correct answer jeopardizes confidentiality of sensitive information, then, instead of the correct answer, either a *lie* or a *refusal* is returned.

CQE methods come in various types and forms depending on the value of three parameters used to specify the semantics of the *confidentiality policies*, the *user awareness* and the *enforcement policies* [4, 5]. Confidentiality policies

include *secrecies* and *potential secrets*. Roughly speaking, when a method is devised to protect secrecies, then the truth value of a secret may not be revealed or inferred. On the other hand, when a method protects potential secrets, one of the two possible truth values is designated as secret, and the method is charged with not revealing that value or allowing the user to infer that value. Its negation, however, is not protected. User awareness is the parameter that specifies whether or not the user is aware of the information whose truth value the knowledge base is trying to conceal. Finally, the enforcement policies that might be employed to ensure preservation of confidentiality are *lying*, *refusal* or a *combination* of lying and refusal.

CQE was first introduced by Sicherman et al. [14] for the method of refusal for protecting known and unknown secrecies. Bonatti et al. [9] discuss the case of lying for known potential secrets. Lying and refusal for known and unknown secrecies is taken up in [2]. Lying and refusal for known potential secrets is studied in [3]. Finally, Biskup and Bonatti explore a combination of refusal and lying for known secrecies and known potential secrets in [6]. Related later works include the creation of a SAT-based algorithm to pre-process information to create an inference-proof database that can be queried statically [10], as well as using constraint satisfaction for the construction of a secure database that also allows static querying without disclosing sensitive information [7]. It is worth noting that, recently, Biskup and Weibert have extended aspects of CQE to the setting of incomplete databases [8].

In controlled query evaluation, a central theme is the tradeoff between confidentiality of secret information and availability of information (see, e.g., the introduction in [10]). In fact, a recurring issue in all aforementioned works on CQE is a comparison of the various enforcement methods with respect to availability. Many take the forms of "honeymoon lemmas" comparing answers provided to initial segments of incoming query sequences [6]. Others are based on rearranging the incoming query sequence to achieve comparability (see, also, [6]). It seems that order, which is indispensable in an algorithmic treatment of security in information management, is presenting a hinderance when trying to compare at a theoretical level the various enforcement methods with respect to availability. In this paper, we make an attempt at a cleaner approach to this issue, based on an unordered view of confidentiality preserving query evaluation.

Our contribution is two-fold. First, we provide the definition of *unordered query evaluation*. Controlled query evaluation is usually presented in two levels of abstraction. In the higher level, a *declarative framework* is introduced, where a description of the method and its confidentiality requirements are given. In the lower level of abstraction, a detailed *algorithmic procedure* is provided for enforcing the confidentiality goals specified at the declarative level. We view unordered query evaluation as a third (the highest) level of abstraction, which "forgets" the order in which the queries are handled and considers only the association with each query of the corresponding response returned to the user. In this setting the two lower levels are viewed as describing a specific possible "order-dependent" implementation strategy for unordered query evaluation. We

use the unordered query evaluation framework to provide a setting in which to estimate the *relative availability* of confidentiality preserving query answering. By associating an *unordered query evaluation companion* to each given CQE method, we carry the applicability of this comparison framework to CQE. We use a comparison result of Biskup and Bonatti [6] to illustrate how our definition may be used to accommodate results of similar kind presented previously in the literature.

The second contribution of the paper is the theoretical *characterization of the maximally available unordered query evaluation methods*. This result also relies on the construction of an unordered query evaluation related to a given CQE. We show that an unordered query evaluation is maximally available iff, for every knowledge base and confidential information, there exists an appropriate ordering of all possible queries, such that the given unordered query evaluation provides identical answers with a CQE processing the queries in the devised order.

The paper is organized as follows. In Section 2 we review the basic definitions and the formalism pertaining to the confidentiality requirements of CQE. In Section 3, we introduce unordered query evaluation and present its confidentiality requirements. Unordered query evaluation is more abstract than CQE, lacking the algorithmic flavor of CQE that results from dealing with the queries in the order in which they are posed to the system. In Section 4, we introduce a relative measure of availability in unordered query evaluation. Moreover, we formalize a way in which an unordered query evaluation may be canonically associated with a CQE and a given ordering of all queries. By using this unordered companion to the CQE and the availability comparison framework for the unordered case, we obtain a relative measure of availability for CQE. In Section 5 we present an availability comparison result of Biskup and Bonatti for various enforcement policies in the case of known potential secrets, placing it in the framework of the present paper as an illustration of the applicability of our general definitions. Finally, in Section 6 we provide a characterization of maximally available unordered query evaluation in terms of the known algorithms for the various enforcement policies in CQE. We concentrate on known potential secrets, but our method is general enough to encompass unknown potential secrets as well as known and unknown secrecies.

2 Controlled Query Evaluation

2.1 Basic Definitions

An *information system* [4, 5] consists of two pieces of data: First, a *schema* DS, which captures the universe of discourse of an intended application and which, for the purposes of this note, will be a finite set of propositional variables. Second, an instance db, which, in general, is a structure interpreting the symbols in DS and, which, for the purposes of this note, will be an assignment of truth values (true ($\mathbf{t}$) or false ($\mathbf{f}$)) to the propositional variables in DS. A *query Φ against* DS is a sentence in classical propositional logic with variables in DS. A *query*

evaluation eval(Φ) determines the truth value of a query Φ against the schema DS for the current instance db as follows:

$$\text{eval}(\Phi) : \text{DS} \to \{\mathbf{t}, \mathbf{f}\} \text{ with } \text{eval}(\Phi)(\text{db}) = \text{db } \texttt{model\_of } \Phi,$$

where `model_of` is the boolean operator returning $\mathbf{t}$ iff db is a model of Φ in the ordinary sense. As is customary, we also use another version eval*, that returns either the query sentence or its negation:

$$\text{eval}^*(\Phi) : \text{DS} \to \{\Phi, \neg\Phi\}, \quad \text{with}$$

$$\text{eval}^*(\Phi)(\text{db}) = \begin{cases} \Phi, & \text{if db } \texttt{model\_of } \Phi \\ \neg\Phi, & \text{otherwise} \end{cases}.$$

Let $Q = \langle \Phi_1, \Phi_2, \ldots, \Phi_i, \ldots \rangle$ be a (possibly infinite) query sequence and $\log_0$ be an *initial user log*, which represents the explicit part of the user's *assumed knowledge*. We define a *controlled query evaluation* as a family of partial functions control_eval$(Q, \log_0)$, each of which has as parameters the query sequence Q and the initial user log $\log_0$. The inputs are "admissible" pairs $(\text{db}, \text{policy})$, where db is an instance of the information system and policy an instance of a confidentiality policy, which can be a set of *secrecies* or a set of *potential secrets*, as in [4, 5]. Admissibility of $(\text{db}, \text{policy})$ is determined by some formally defined precondition precond associated with the function.

For any specific CQE function, the choices with respect to model of policy (secrecies or potential secrets), user awareness (unknown or known policy) and enforcement method (lying, refusal or combined) are indicated by attaching the superscripts p, a, e, with $p \in \{\texttt{sec}, \texttt{ps}\}, a \in \{\texttt{unknown}, \texttt{known}\}$ and $e \in \{\texttt{L(ying)}, \texttt{R(efusal)}, \texttt{C(ombined)}\}$. In specifying such a function, it is assumed that, given a query, the correct answer to the query according to the current database instance is judged by some *censor*, which decides whether the correct answer may be disclosed or whether a *modificator* must be applied. The censor is assisted by a *user log* log, which represents the explicit part of the user's *assumed knowledge*, much like $\log_0$ represents the explicit part of the user's initial assumed knowledge. The log is updated every time an answer is returned. Therefore, the function is given by

$$\text{control\_eval}^{p,a,e}(Q, \log_0)(\text{db}, \text{policy}) = \langle (\text{ans}_1, \log_1), \ldots, (\text{ans}_i, \log_i), \ldots \rangle,$$

where $\log_i = \begin{cases} \log_{i-1}, & \text{if ans}_i = \texttt{mum} \\ \log_{i-1} \cup \{\text{ans}_i\}, & \text{otherwise} \end{cases}$, `mum` signifying refusal to answer.

2.2 Confidentiality Requirements

Depending on whether the model of confidentiality policy is that of secrecies or of potential secrets, we have appropriately adjusted instances of the policy. For the model of secrecies, we have a finite set secr $= \{\{\Psi_1, \neg\Psi_1\}, \ldots, \{\Psi_k, \neg\Psi_k\}\}$ of complementary pairs of sentences, each called a *secrecy*. On the other hand, for

potential secrets, we have a finite set pot_sec $= \{\Psi_1, \ldots, \Psi_k\}$ of sentences, called *potential secrets*. The semantics for a secrecy $\{\Psi, \neg\Psi\}$ requires that a user should not be able to distinguish, based on initial knowledge and answers returned by the system, whether Ψ or $\neg\Psi$ is true in the actual instance of the information system. For a potential secret Ψ, a user should not be able to exclude that $\neg\Psi$ is true in the actual instance of the information system. More formally, we have the following definition for preservation of confidentiality for a given controlled query evaluation function (see Definition 1 of [4]):

Definition 1. *Let* control_eval$^{p,a,e}(Q, \log_0)$ *be a specific controlled query evaluation with* precond *as its associated precondition and* policy$_1$ *a policy instance.*

1. control_eval$^{p,a,e}(Q, \log_0)$ *is said to* preserve confidentiality with respect to policy$_1$ *iff, for all finite prefixes Q' of Q, all instances* db$_1$ *of the information system, such that* (db$_1$, policy$_1$) *satisfies* precond *and all $\Theta \in$* policy$_1$*, there exists* db$_2$ *and* policy$_2$*, such that* (db$_2$, policy$_2$) *satisfies* precond *and the following conditions hold:*
   (a) [Same Answers] control_eval$^{p,a,e}(Q', \log_0)$(db$_1$, policy$_1$) = control_eval$^{p,a,e}(Q', \log_0)$(db$_2$, policy$_2$)
   (b) [Different Secrets/False Potential Secrets] *If* $p =$ `sec`*, i.e.,* $\Theta = \{\Psi, \neg\Psi\}$ *a secrecy,* $\{\text{eval}^*(\Psi)(\text{db}_1), \text{eval}^*(\Psi)(\text{db}_2)\} = \{\Psi, \neg\Psi\}$ *and, if* $p =$ `ps`*, i.e.,* $\Theta = \Psi$ *is a potential secret, then* $\text{eval}^*(\Psi)(\text{db}_2) = \neg\Psi$
   (c) [Awareness] *if* $a =$ `known`*, then* policy$_1$ = policy$_2$.
2. control_eval$^{p,a,e}(Q, \log_0)$ *is said to* preserve confidentiality *if it preserves confidentiality with respect to all admissible policy instances.*

3 Unordered Query Evaluation

In this section, we introduce *unordered query evaluation*, which is a modification of controlled query evaluation in which the ordering of the queries does not play a role. It is closer in spirit to alternative treatments of secrecy preserving reasoning that have been introduced in the literature, namely those by Studder [16] (see also [15, 17]) and Bao et al. [1]. The reason for defining unordered query evaluation is that it provides a more elegant way to compare methods of dynamic enforcement of controlled query evaluation with respect to their availability, which is the main topic of this paper. The way this can be achieved will be illustrated in Sections 5 and 6. In this section we give the definition and the relevant confidentiality requirements.

We adopt the same notion of *information system* that was used in Section 2 and the same notion of query and query evaluation. An *unordered query evaluation* is a family of partial functions unord_eval($\log_0$), each of which has as a parameter the initial user log $\log_0$. The inputs are "admissible" pairs (db, policy), where db is an instance of the information system and policy an instance of a confidentiality policy, which, as before, can be a set of *secrecies* or a set of *potential secrets*. A precondition precond determines the admissibility of

(db, policy). The superscripts p, a, e, with $p \in \{\texttt{sec}, \texttt{ps}\}$, $a \in \{\texttt{unknown}, \texttt{known}\}$ and $e \in \{\texttt{L(ying)}, \texttt{R(efusal)}, \texttt{C(ombined)}\}$, are used, as before, to indicate model of policy, user awareness and method of enforcement, respectively.

unord_eval$^{p,a,e}(\log_0)(\text{db}, \text{policy})$ is a total function from the set of queries $\mathcal{Q}$ to the set of possible answers, i.e., for all $\Phi \in \mathcal{Q}$,

$$\text{unord\_eval}^{p,a,e}(\log_0)(\text{db}, \text{policy})(\Phi) \in \{\Phi, \neg\Phi, \texttt{mum}\}.$$

The following definition for preservation of confidentiality for a given unordered query evaluation function adapts the corresponding one for controlled query evaluation (Definition 1):

Definition 2. *Let* unord_eval$^{p,a,e}(\log_0)$ *be a specific unordered query evaluation with* precond *as its associated precondition and* policy_1 *a policy instance.*

1. unord_eval$^{p,a,e}(\log_0)$ *is said to* preserve confidentiality with respect to policy_1 *iff, for every finite* $\mathcal{Q}' \subseteq \mathcal{Q}$*, all instances* db_1 *of the information system, such that* $(\text{db}_1, \text{policy}_1)$ *satisfies* precond *and all* $\Theta \in \text{policy}_1$*, there exists* db_2 *and* policy_2*, such that* $(\text{db}_2, \text{policy}_2)$ *satisfies* precond *and the following conditions hold:*
   - *(a)* [Same Answers] *for all* $\Phi \in \log_0 \cup \mathcal{Q}'$, unord_eval$^{p,a,e}(\log_0)(\text{db}_1, \text{policy}_1)(\Phi) = \text{unord_eval}^{p,a,e}(\log_0)(\text{db}_2, \text{policy}_2)(\Phi)$
   - *(b)* [Different Secrets/False Potential Secrets] *If* $p = \texttt{sec}$*, i.e.,* $\Theta = \{\Psi, \neg\Psi\}$ *a secrecy,* $\{\text{eval}^*(\Psi)(\text{db}_1), \text{eval}^*(\Psi)(\text{db}_2)\} = \{\Psi, \neg\Psi\}$ *and, if* $p = \texttt{ps}$*, i.e.,* $\Theta = \Psi$ *is a potential secret, then* $\text{eval}^*(\Psi)(\text{db}_2) = \neg\Psi$
   - *(c)* [Awareness] *if* $a = \texttt{known}$*, then* $\text{policy}_1 = \text{policy}_2$.
2. unord_eval$^{p,a,e}(\log_0)$ *is said to* preserve confidentiality *if it preserves confidentiality with respect to all admissible policy instances.*

4 Availability

4.1 Availability in Unordered Query Evaluation

In many instances in previous work on controlled query evaluation, there has been explicit reference to the important tradeoff between *confidentiality* of secret information and *availability* of information (see, e.g., [10]). In general, in the framework of CQE, one way that has been employed for comparisons of difference enforcement methods with respect to availability has been via the so-called "Honeymoon Lemmas" and, also, via some query reordering-style lemmas [6]. In this subsection, we provide a general definition for comparing the availability of two *unordered* query evaluations. In the following subsection, we will show how a controlled query evaluation gives rise to an *unordered query evaluation companion* (essentially by forgetting the order of the queries) and we will use this associated unordered query evaluation together with the comparison framework of this subsection to provide a setting for comparing controlled query evaluations with respect to availability. Finally, in Section 5 we show how a comparison lemma of Biskup and Bonatti can be seen as a particular comparison result on availability in the sense of the present section.

Given a database instance db and a query Φ, we define a partial ordering $\leq_{\mathrm{db}}$ on the set $\{\Phi, \neg\Phi, \mathtt{mum}\}$ of the three possible answers of a controlled query evaluation on Φ by setting

$$\mathtt{mum} \leq_{\mathrm{db}} \mathrm{eval}^*(\Phi)(\mathrm{db}), \quad \neg\mathrm{eval}^*(\Phi)(\mathrm{db}) \leq_{\mathrm{db}} \mathrm{eval}^*(\Phi)(\mathrm{db}).$$

Let $\mathrm{unord\_eval}_1^{p,a,e_1}(\log_0)$ and $\mathrm{unord\_eval}_2^{p,a,e_2}(\log_0)$ be unordered query evaluations, with associated preconditions $\mathrm{precond}_1$ and $\mathrm{precond}_2$, respectively, and $(\mathrm{db}, \mathrm{policy})$ an admissible pair according to both $\mathrm{precond}_1$ and $\mathrm{precond}_2$. Then

$$\mathrm{unord\_eval}_1^{p,a,e_1}(\log_0)(\mathrm{db}, \mathrm{policy}) \leq \mathrm{unord\_eval}_2^{p,a,e_2}(\log_0)(\mathrm{db}, \mathrm{policy})$$

signifies that, for all $\Phi \in \mathcal{Q}$ (the set of all queries),

$$\mathrm{unord\_eval}_1^{p,a,e_1}(\log_0)(\mathrm{db}, \mathrm{policy})(\Phi) \leq_{\mathrm{db}} \mathrm{unord\_eval}_2^{p,a,e_2}(\log_0)(\mathrm{db}, \mathrm{policy})(\Phi).$$

Definition 3. *Let* $\mathrm{unord\_eval}_1^{p,a,e_1}$ *and* $\mathrm{unord\_eval}_2^{p,a,e_2}$ *be unordered query evaluations with preconditions* $\mathrm{precond}_1$ *and* $\mathrm{precond}_2$*, respectively.*

1. $\mathrm{unord\_eval}_1^{p,a,e_1}$ *is said to be* more available than $\mathrm{unord\_eval}_2^{p,a,e_2}$ with respect to *assumed knowledge* $\log_0$ and *confidentiality policy* policy *if, for every database instance* db*, such that* $(\mathrm{db}, \mathrm{policy})$ *is admissible according to both* $\mathrm{precond}_1$ *and* $\mathrm{precond}_2$*, we have that*

$$\mathrm{unord\_eval}_1^{p,a,e_1}(\log_0)(\mathrm{db}, \mathrm{policy}) \geq \mathrm{unord\_eval}_2^{p,a,e_2}(\log_0)(\mathrm{db}, \mathrm{policy});$$

2. $\mathrm{unord\_eval}_1^{p,a,e_1}$ *is said to be* maximally available with respect to *assumed knowledge* $\log_0$ and *confidentiality policy* policy *if, for every unordered query evaluation* $\mathrm{unord\_eval}_2^{p,a,e_1}$*, all database instances* db*, such that* $(\mathrm{db}, \mathrm{policy})$ *is admissible according to both* $\mathrm{precond}_1$ *and* $\mathrm{precond}_2$*, we have that*

$$\mathrm{unord\_eval}_1^{p,a,e_1}(\log_0)(\mathrm{db}, \mathrm{policy}) \not< \mathrm{unord\_eval}_2^{p,a,e_1}(\log_0)(\mathrm{db}, \mathrm{policy}).$$

4.2 From Controlled to Unordered Query Evaluation

In this subsection we show how to construct, given a controlled query evaluation, an associated unordered query evaluation, called its *unordered query evaluation companion*. We use the construction for the purpose of rigorously comparing controlled query evaluations with respect to availability.

Suppose that $\mathrm{control\_eval}^{p,a,e}$ is a controlled query evaluation, $\log_0$ a user's initial assumed knowledge and Q an infinite sequence, which is surjective on $\mathcal{Q}$, i.e., whose range includes all possible queries against the schema DS. Define the *unordered query evaluation companion* $\mathrm{unord\_eval}^{p,a,e}(\log_0)$ *of* $\mathrm{control\_eval}^{p,a,e}$ *relative to* $\log_0$ *and* Q as follows:

The precondition precond of the unordered query evaluation includes all pairs $(\mathrm{db}, \mathrm{policy})$ that are included in the precondition for $\mathrm{control\_eval}^{p,a,e}(Q, \log_0)$. Moreover, for all pairs $(\mathrm{db}, \mathrm{policy}) \in \mathrm{precond}$ and all $\Phi \in \mathcal{Q}$, we define

$$\mathrm{unord\_eval}^{p,a,e}(\log_0)(\mathrm{db}, \mathrm{policy})(\Phi) = \mathrm{ans}_i,$$

where

$$\text{control\_eval}^{p,a,e}(Q, \log_0)(\text{db}, \text{policy}) = \langle(\text{ans}_1, \log_1), \ldots, (\text{ans}_i, \log_i), \ldots\rangle$$

and i is the first occurrence of Φ in the sequence $Q = \langle\Phi_1, \Phi_2, \ldots, \Phi_i, \ldots\rangle$.

Proposition 1. *If a controlled query evaluation* $\text{control\_eval}^{p,a,e}(Q, \log_0)$ *preserves confidentiality with respect to* policy*, then its unordered query evaluation companion* $\text{unord\_eval}^{p,a,e}(\log_0)$ *relative to* $\log_0$ *and* Q *also preserves confidentiality with respect to* policy*.*

Proof. (Sketch) Suppose $\text{control\_eval}^{p,a,e}(Q, \log_0)$, with precondition precond and policy_1 a policy instance, is a CQE, that preserves confidentiality with respect to policy_1. Let $\text{unord\_eval}^{p,a,e}(\log_0)$ be its unordered query evaluation companion relative to $\log_0$ and Q. Consider finite $\mathcal{Q}' \subseteq \mathcal{Q}$ and a database instance db_1, such that $(\text{db}_1, \text{policy}_1) \in \text{precond}$, and $\Theta \in \text{policy}_1$. Let Q' be a finite prefix of Q, whose range includes $\mathcal{Q}'$. Since precond is a common precondition for both $\text{control\_eval}^{p,a,e}(Q, \log_0)$ and $\text{unord\_eval}^{p,a,e}(\log_0)$, and $(\text{db}_1, \text{policy}_1) \in \text{precond}$, there exists, by preservation of confidentiality for the CQE, $(\text{db}_2, \text{policy}_2) \in \text{precond}$, such that all three conditions of Definition 1 are satisfied. Now it is straightforward to check that all three corresponding conditions of Definition 2 hold for $\text{unord\_eval}^{p,a,e}(\log_0)$ and, therefore, $\text{unord\_eval}^{p,a,e}(\log_0)$ also preserves confidentiality.

The definitions for comparing availability for unordered query evaluations may now be applied to the case of controlled query evaluations using their unordered query evaluation companions.

Definition 4. *Let* $\text{control\_eval}_1^{p,a,e_1}$ *and* $\text{control\_eval}_2^{p,a,e_2}$ *be controlled query evaluations with preconditions* precond_1 *and* precond_2*, respectively,* $\log_0$ *a user's initial assumed knowledge and* Q_1, Q_2 *infinite sequences, which are surjective on* $\mathcal{Q}$*.*

1. $\text{control\_eval}_1^{p,a,e_1}(Q_1, \log_0)$ *is said to be* more available than $\text{control\_eval}_2^{p,a,e_2}(Q_2, \log_0)$ with respect to *confidentiality policy* policy *if* $\text{unord\_eval}_1^{p,a,e_1}$ *is more available than* $\text{unord\_eval}_2^{p,a,e_2}$ *with respect to assumed knowledge* $\log_0$ *and confidentiality policy* policy*, where*

$$\text{unord\_eval}_1^{p,a,e_1}(\log_0) \quad \textit{and} \quad \text{unord\_eval}_2^{p,a,e_2}(\log_0)$$

   are the unordered query evaluation companions of $\text{control\_eval}_1^{p,a,e_1}(Q_1, \log_0)$ *and* $\text{control\_eval}_2^{p,a,e_2}(Q_2, \log_0)$*, respectively.*
2. $\text{control\_eval}_1^{p,a,e_1}(Q_1, \log_0)$ *is said to be* maximally available with respect to *confidentiality policy* policy *if* $\text{unord\_eval}_1^{p,a,e_1}$ *is maximally available with respect to assumed knowledge* $\log_0$ *and confidentiality policy* policy*.*

It is worth noting that in Definition 4 we chose to define maximal availability for controlled query evaluations to mean that the corresponding unordered query evaluation is the "best" among all other unordered query evaluations having the *same enforcement policy.*

5 A Comparison Result of Biskup and Bonatti

In this section, we revisit a result formulated and proven by Biskup and Bonatti in [6] on comparing the uniform with the hybrid methods in the case of known potential secrets in order to illustrate how this and other results of similar flavor may be accommodated in the general comparison framework that was presented in the previous section.

5.1 Uniform Refusal and Lying for Known Potential Secrets

We start by briefly reviewing the definition of the *refusal censor* and the resulting controlled query evaluation procedure in the case of *known potential secrets*. The requirement is that the censor refuse the answer to a query if the correct answer or its negation together with the current log imply a potential secret:

$$\begin{aligned}\text{censor}^{\mathtt{R}}_{\mathtt{ps}}(\Phi, \log, \text{db}, \text{pot\_sec}) = (\exists \Psi)[\Psi \in \text{pot\_sec} \text{ and } (\log \cup \{\text{eval}^*(\Phi)(\text{db})\} \models \Psi \\ \text{or } \log \cup \{\neg\text{eval}^*(\Phi)(\text{db})\} \models \Psi)].\end{aligned}$$

Now, $\text{control\_eval}^{\mathtt{R}}_{\mathtt{ps}}(Q, \log_0)(\text{db}, \text{pot\_sec})$ is defined by

$$\text{ans}_i = \begin{cases} \mathtt{mum}, & \text{if } \text{censor}^{\mathtt{R}}_{\mathtt{ps}}(\Phi_i, \log_{i-1}, \text{db}, \text{pot\_sec}) \\ \text{eval}^*(\Phi_i)(\text{db}), & \text{otherwise} \end{cases}$$

$$\log_i = \begin{cases} \log_{i-1}, & \text{if } \text{ans}_i = \mathtt{mum} \\ \log_{i-1} \cup \{\text{ans}_i\}, & \text{otherwise} \end{cases}$$

In the case of refusal the appropriate precondition for applying the algorithm is $(\text{db}, \text{policy}) \in \text{precond}$ iff db `model_of` $\log_0$.

We next review the definition of the *lying censor* and the resulting controlled query evaluation procedure in the case of *known potential secrets*. The requirement is that the censor refuse the answer to a query if the correct answer together with the current log imply the *disjunction of all potential secrets*:

$$\begin{aligned}&\text{censor}^{\mathtt{L}}_{\mathtt{ps}}(\Phi, \log, \text{db}, \text{pot\_sec}) = \log \cup \{\text{eval}^*(\Phi)(\text{db})\} \models \text{pot\_sec\_disj} \\ &\text{where } \text{pot\_sec\_disj} = \textstyle\bigvee_{\Psi \in \text{pot\_sec}} \Psi.\end{aligned}$$

Now, $\text{control\_eval}^{\mathtt{L}}_{\mathtt{ps}}(Q, \log_0)(\text{db}, \text{pot\_sec})$ is defined by

$$\text{ans}_i = \begin{cases} \neg\text{eval}^*(\Phi_i)(\text{db}), & \text{if } \text{censor}^{\mathtt{L}}_{\mathtt{ps}}(\Phi_i, \log_{i-1}, \text{db}, \text{pot\_sec}) \\ \text{eval}^*(\Phi_i)(\text{db}), & \text{otherwise} \end{cases}$$

$$\log_i = \log_{i-1} \cup \{\text{ans}_i\}.$$

In the case of lying the appropriate precondition for applying the algorithm is that for all $\Psi \in \text{pot\_sec}$, $\log_0 \not\models \Psi$.

5.2 Combined Method for Known Potential Secrets

Biskup and Bonatti [6], realizing that both uniform controlled query evaluation methods (Refusal and Lying) have disadvantages, introduced the *combined lying and refusal method* for *known potential secrets*. In this method, when a query Φ is posed, the database could refuse, lie or provide the correct answer. Refusal occurs when the current log and the correct answer imply a potential secret and, in addition, the current log and the false answer also imply a potential secret. Lying occurs when the current log and the correct answer imply a potential secret but the current log and the false answer do not. Finally, the correct answer is provided in case the current log and the correct answer do not imply any potential secrets.

Thus, the controlled query evaluation method by combining refusal and lying for known potential secrets, $\text{control\_eval}^{\mathtt{C}}_{\mathtt{ps}}(Q, \log_0)(\text{db}, \text{pot\_sec})$, with

$$\text{control\_eval}^{\mathtt{C}}_{\mathtt{ps}}(Q, \log_0)(\text{db}, \text{pot\_sec}) = \langle(\text{ans}_1, \log_1), \ldots, (\text{ans}_i, \log_i), \ldots\rangle,$$

is defined by

$$\text{ans}_i = \begin{cases} \mathtt{mum}, & \text{if } (\exists \Psi_1, \Psi_2 \in \text{pot\_sec})(\log_{i-1} \cup \{\text{eval}^*(\Phi_i)(\text{db})\} \models \Psi_1 \\ & \qquad \text{and } \log_{i-1} \cup \{\neg\text{eval}^*(\Phi_i)(\text{db})\} \models \Psi_2) \\ \neg\text{eval}^*(\Phi_i)(\text{db}), & \text{if } (\exists \Psi_1 \in \text{pot\_sec})(\log_{i-1} \cup \{\text{eval}^*(\Phi_i)(\text{db})\} \models \Psi_1) \\ & \text{and } (\nexists \Psi_2 \in \text{pot\_sec})(\log_{i-1} \cup \{\neg\text{eval}^*(\Phi_i)(\text{db})\} \models \Psi_2) \\ \text{eval}^*(\Phi_i)(\text{db}), & \text{otherwise} \end{cases}$$

$$\log_i = \begin{cases} \log_{i-1}, & \text{if } \text{ans}_i = \mathtt{mum} \\ \log_{i-1} \cup \{\text{ans}_i\}, & \text{otherwise} \end{cases}.$$

In the case of the combined method the appropriate precondition for applying the algorithm is, for all $\Psi \in \text{pot\_sec}$, $\log_0 \not\models \Psi$. In Theorem 1 of [6], it is shown that the combined method is secure according to the general Definition 1.

5.3 Comparison Result

In Theorem 2 of [6], Biskup and Bonatti prove that the combined method is "more cooperative" than any of the uniform methods for *known potential secrets*. In Section 5 of [6], they also point out that the same holds for the case of *known secrecies*. In this section, after revisiting their result, we see that the informal notion of "more cooperative" is accurately captured by our formal notion of "more available", as detailed in Definition 4.

Theorem 1 (Biskup and Bonatti). *Let* $\mathtt{M}$ *denote either refusal (*$\mathtt{R}$*) or lying (*$\mathtt{L}$*) and* $\log_0$, $(\text{db}, \text{pot\_sec})$ *appropriate parameters. Then, for all query sequences* Q^M *for uniform* $\mathtt{M}$*, there exists a query sequence* Q^C *for the combined method, such that:*

1. $\text{control\_eval}^{\mathtt{C}}_{\mathtt{ps}}(Q^C, \log_0)(\text{db}, \text{pot\_sec})$ *delivers all the answers that are correct under* $\text{control\_eval}^{\mathtt{M}}_{\mathtt{ps}}(Q^M, \log_0)(\text{db}, \text{pot\_sec})$ *and possibly more correct answers.*

2. *Q^C is defined by a reordering that shifts correctly answered queries towards the beginning of the query sequence.*

Using the terminology introduced in the present paper, Theorem 1 may be rephrased as follows:

Theorem 2. *Let* $\mathtt{M}$ *denote either refusal (*$\mathtt{R}$*) or lying (*$\mathtt{L}$*) and* $\log_0$, $(\mathrm{db}, \mathrm{pot\_sec})$ *appropriate parameters. Then, for all query sequences Q^M for uniform* $\mathtt{M}$*, that are surjective on $\mathcal{Q}$, there exists a query sequence Q^C for the combined method, also surjective on $\mathcal{Q}$, such that* $\mathrm{control\_eval}^{\mathtt{C}}_{\mathtt{ps}}(Q^C, \log_0)$ *is more available with respect to* pot_sec *than* $\mathrm{control\_eval}^{\mathtt{M}}_{\mathtt{ps}}(Q^M, \log_0)$

Theorem 2 is a direct consequence of Theorem 1 and Definition 4.

6 Maximal Availability for Known Potential Secrets

Since, in Definition 4 of maximal availability for a controlled query evaluation, its corresponding unordered query evaluation is compared with other unordered query evaluations having the *same enforcement policy*, Theorem 2 does not have any impact on maximality, since it is a comparison between controlled query evaluations with different enforcement policies. In this section, we undertake the task of showing that all three enforcement policies for known potential secrets are maximally available. We also present a result connecting maximally available unordered query evaluation with CQE. This result is related to the reordering of query sequences that has been a recurring theme in the studies of controlled query evaluation for various enforcement policies by Biskup and Bonatti (see, e.g., Lemma 2 of [6]), as well as with the, so-called, *order-induced secrecy preserving reasoners* studied in [18].

Theorem 3. *Let* $\mathtt{M}$ *denote refusal (*$\mathtt{R}$*), lying (*$\mathtt{L}$*) or the combined (*$\mathtt{C}$*) enforcement method,* $\log_0$, $(\mathrm{db}, \mathrm{pot\_sec})$ *appropriate parameters and $Q = \langle \Phi_1, \Phi_2, \ldots, \Phi_i, \ldots \rangle$ a query sequence that is surjective on $\mathcal{Q}$. Then,* $\mathrm{control\_eval}^{\mathtt{M}}_{\mathtt{ps}}(Q, \log_0)$ *is maximally available with respect to* pot_sec*.*

Proof. Let $\mathrm{unord\_eval}^{\mathtt{M}}_{\mathtt{ps}}(\log_0)(\mathrm{db}, \mathrm{pot\_sec})$ be the unordered query evaluation companion of $\mathrm{control\_eval}^{\mathtt{M}}_{\mathtt{ps}}(Q, \log_0)$. Suppose, for the sake of obtaining a contradiction, that $\mathrm{control\_eval}^{\mathtt{M}}_{\mathtt{ps}}(Q, \log_0)$ is not maximally available with respect to pot_sec. Thus, there exists $\mathrm{unord\_eval}'^{\mathtt{M}}_{\mathtt{ps}}(\log_0)(\mathrm{db}, \mathrm{pot\_sec})$ and db', such that $(\mathrm{db}', \mathrm{pot\_sec})$ is admissible for both unordered query evaluations and such that $\mathrm{unord\_eval}^{M}_{\mathtt{ps}}(\log_0)(\mathrm{db}', \mathrm{pot\_sec}) < \mathrm{unord\_eval}'^{M}_{\mathtt{ps}}(\log_0)(\mathrm{db}', \mathrm{pot\_sec})$. This means that there exists $i \geq 1$, such that

$$\mathrm{unord\_eval}^{\mathtt{M}}_{\mathtt{ps}}(\log_0)(\mathrm{db}', \mathrm{pot\_sec})(\Phi_i) <_{\mathrm{db}'} \mathrm{unord\_eval}'^{\mathtt{M}}_{\mathtt{ps}}(\log_0)(\mathrm{db}', \mathrm{pot\_sec})(\Phi_i).$$

Consider the smallest such i. Then, we must have, for all $j < i$,

$$\mathrm{unord\_eval}^{\mathtt{M}}_{\mathtt{ps}}(\log_0)(\mathrm{db}', \mathrm{pot\_sec})(\Phi_j) = \mathrm{unord\_eval}'^{\mathtt{M}}_{\mathtt{ps}}(\log_0)(\mathrm{db}', \mathrm{pot\_sec})(\Phi_j)$$

and $\text{unord\_eval}'^{\mathtt{M}}_{\mathtt{ps}}(\log_0)(\text{db}', \text{pot\_sec})(\Phi_i) <_{\text{db}'} \text{eval}^*(\Phi_i)(\text{db}')$. But, then, using the definition of $\text{unord\_eval}^{\mathtt{M}}_{\mathtt{ps}}(\log_0)(\text{db}', \text{pot\_sec})$ together with the description of $\text{control\_eval}^{\mathtt{M}}_{\mathtt{ps}}(Q, \log_0)$, we conclude that $\text{unord\_eval}'^{\mathtt{M}}_{\mathtt{ps}}(\log_0)$ does not preserve confidentiality, which is a contradiction. Thus, $\text{control\_eval}^{\mathtt{M}}_{\mathtt{ps}}(Q, \log_0)$ is maximally available.

Theorem 4. *Let* $\mathtt{M}$ *denote refusal (*$\mathtt{R}$*), lying (*$\mathtt{L}$*) or the combined (*$\mathtt{C}$*) enforcement method and* $\log_0$, $(\text{db}, \text{pot\_sec})$ *appropriate parameters for a maximally available unordered query evaluation* $\text{unord\_eval}^{\mathtt{M}}_{\mathtt{ps}}(\log_0)$*. Then, there exists a query sequence* $Q = \langle \Phi_1, \Phi_2, \ldots, \Phi_i, \ldots \rangle$*, that is surjective on* $\mathcal{Q}$*, such that the controlled query evaluation* $\text{control\_eval}^{\mathtt{M}}_{\mathtt{ps}}(Q, \log_0)$*, with precondition* precond*, satisfies*

1. $(\text{db}, \text{pot\_sec}) \in \text{precond}$ *and*
2. $\text{control\_eval}^{\mathtt{M}}_{\mathtt{ps}}(Q, \log_0)(\text{db}, \text{pot\_sec}) = \langle (\text{ans}_1, \log_1), \ldots, (\text{ans}_i, \log_i), \ldots \rangle$*, with* $\text{ans}_i = \text{unord\_eval}^{\mathtt{M}}_{\mathtt{ps}}(\log_0)(\text{db}, \text{pot\_sec})(\Phi_i)$*, for all* $i \geq 1$*.*

Proof. We present the proof for $\mathtt{M} = \mathtt{L}$, i.e., for the uniform lying enforcement policy. The proofs for the other two cases are similar.

Let $\text{unord\_eval}^{\mathtt{L}}_{\mathtt{ps}}(\log_0)$ be a maximally available unordered query evaluation, $(\text{db}, \text{pot\_sec})$ appropriate parameters, $\text{Acc} = \{\Gamma_1, \Gamma_2, \ldots\}$ be the set of all accurate answers, i.e., such that $\text{unord\_eval}^{\mathtt{L}}_{\mathtt{ps}}(\log_0)(\text{db}, \text{pot\_sec})(\Gamma_i) = \text{eval}^*(\Gamma_i)(\text{db})$, for all i, and $\text{Alt} = \{\Delta_1, \Delta_2, \ldots\}$ the set of all altered answers, i.e., such that $\text{unord\_eval}^{\mathtt{L}}_{\mathtt{ps}}(\log_0)(\text{db}, \text{pot\_sec})(\Delta_i) = \neg\text{eval}^*(\Delta_i)(\text{db})$, for all i. To shorten notation, we write $\Gamma_i^* = \text{eval}^*(\Gamma_i)(\text{db})$ and, similarly for Δ_i^*. We also set $\text{Acc}^* = \{\Gamma^* : \Gamma \in \text{Acc}\}$ and similarly for Alt^*. We will exhibit a sequence $Q = \langle \Phi_1, \Phi_2, \ldots \rangle$, surjective on the set of all possible queries $\mathcal{Q}$, such that the controlled query evaluation $\text{control\_eval}^{\mathtt{L}}_{\mathtt{ps}}(Q, \log_0)$ satisfies the conditions of the statement.

The difficulty in constructing such a sequence lies in the fact that Acc may be countably infinite. If not, i.e., if $\text{Acc} = \{\Gamma_1, \Gamma_2, \ldots, \Gamma_n\}$ is finite, then the sequence $\Gamma_1, \ldots, \Gamma_n, \Delta_1, \Delta_2, \ldots$ accomplishes our goal. In the case of countably infinite Acc, consider the sequence

$$\Gamma_1, \Gamma_2, \Gamma_3, \ldots. \tag{1}$$

Since $\text{unord\_eval}^{\mathtt{L}}_{\mathtt{ps}}(\log_0)$ preserves confidentiality, we must have that $\log_0 \cup \text{Acc}^* \not\models \Psi$, for every potential secret Ψ. On the other hand, the maximal availability of $\text{unord\_eval}^{\mathtt{L}}_{\mathtt{ps}}(\log_0)(\text{db}, \text{pot\_sec})$ implies that, for every $i = 1, 2, \ldots$, there exists a potential secret Ψ, such that $\log_0 \cup \text{Acc}^* \cup \{\Delta_i^*\} \models \Psi$. To place the elements $\Delta_1, \Delta_2, \ldots$ in the List (1), we work by induction on the index i as follows:

Since Δ_1 is such that $\log_0 \cup \text{Acc}^* \cup \{\Delta_1^*\} \models \Psi_1$, for some potential secret Ψ_1, there exist finitely many $\Gamma_{i_1^1}, \Gamma_{i_2^1}, \ldots, \Gamma_{i_{m_1}^1} \in \text{Acc}$, $i_1^1 < i_2^1 < \cdots < i_{m_1}^1$, such that $\log_0 \cup \{\Gamma_{i_1^1}^*, \ldots, \Gamma_{i_{m_1}^1}^*\} \cup \{\Delta_1^*\} \models \Psi_1$. Let $l = \max\{i : \Gamma_i^* \in \log_0\}$ and $n_1 = \max\{l, i_{m_1}^1\}$. Insert Δ_1 immediately after Γ_{n_1} in the List (1).

Suppose, next that $\Delta_1, \Delta_2, \ldots, \Delta_{k-1}$ have all been placed in appropriate positions in the List (1). We work to place Δ_k. Since Δ_k is such that $\log_0 \cup \text{Acc}^* \cup \{\Delta_k^*\} \models \Psi_k$, for some potential secret Ψ_k, there exist $\Gamma_{i_1^k}, \Gamma_{i_2^k}, \ldots, \Gamma_{i_{m_k}^k} \in$

Acc, $i_1^k < i_2^k < \cdots < i_{m_k}^k$, such that $\log_0 \cup \{\Gamma_{i_1^k}^*, \ldots, \Gamma_{i_{m_k}^k}^*\} \cup \{\Delta_k^*\} \models \Psi_k$. Let $n_k = \max\{l, i_{m_1}^1, \ldots, i_{m_k}^k, n_{k-1}+1\}$. Insert Δ_k immediately after Γ_{n_k} in the List (1).

We will show by induction on the index i that, for all $i = 1, 2, \ldots$, if $\Phi_i = \Gamma_j$, then $\text{ans}_i = \Gamma_j^*$ and, if $\Phi_i = \Delta_j$, then $\text{ans}_i = \neg\Delta_j^*$, where $\text{control\_eval}_{\mathtt{ps}}^{\mathtt{L}}(Q, \log_0)(\text{db}, \text{pot\_sec}) = \langle(\text{ans}_1, \log_1), \ldots, (\text{ans}_i, \log_i), \ldots\rangle$.

For the basis of the induction, let $i = 1$ and consider two cases:

- If $\Phi_1 = \Gamma_1$, then, by preservation of confidentiality, for all $\Psi \in \text{pot\_sec}$, $\log_0 \cup \{\Gamma_1^*\} \not\models \Psi$. Thus, by the algorithm for lying, $\text{ans}_1 = \Gamma_1^* = \text{unord\_eval}_{\mathtt{ps}}^{\mathtt{L}}(\log_0)(\text{db}, \text{pot\_sec})(\Gamma_1)$.
- If $\Phi_1 = \Delta_1$, then, by the construction, exists $\Psi_1 \in \text{pot\_sec}$, $\Delta_1^* \models \Psi_1$. Thus, a fortiori, $\log_0 \cup \{\Delta_1^*\} \models \Psi_1$. Therefore, $\text{ans}_1 = \neg\Delta_1^* = \text{unord\_eval}_{\mathtt{ps}}^{\mathtt{L}}(\log_0)(\text{db}, \text{pot\_sec})(\Delta_1)$.

Assume, now, that for all $i = 1, \ldots, k-1$, we have that $\text{ans}_i = \text{unord\_eval}_{\mathtt{ps}}^{\mathtt{L}}(\log_0)(\text{db}, \text{pot\_sec})(\Phi_i)$. If $\Phi_k = \Gamma_j$, for some j, then by preservation of confidentiality and the induction hypothesis, for all $\Psi \in \text{pot\_sec}$, $\log_0 \cup \{\text{ans}_1, \ldots, \text{ans}_{k-1}\} \cup \{\text{unord\_eval}_{\mathtt{ps}}^{\mathtt{L}}(\log_0)(\text{db}, \text{pot\_sec})(\Gamma_j)\} \not\models \Psi$. Thus, for all $\Psi \in \text{pot\_sec}$, $\log_{k-1} \cup \{\Gamma_j^*\} \not\models \Psi$. Therefore, $\text{ans}_k = \Gamma_j^* = \text{unord\_eval}_{\mathtt{ps}}^{\mathtt{L}}(\log_0)(\text{db}, \text{pot\_sec})(\Gamma_j)$. The case where $\Phi_i = \Delta_j$, for some j, may be handled similarly.

7 Summary

In this paper, we defined *unordered query evaluation* as a way to study the *relative availability* of various *controlled query evaluation enforcement methods*. We first compare availability of unordered query evaluation in a straightforward way and, then, by associating an *unordered query evaluation companion* to each given CQE method, we carry the applicability of this comparison framework to CQE. We used a comparison result of Biskup and Bonatti [6] to illustrate how our definition may be used to accommodate availability comparison results considered in the literature.

We also theoretically connected *maximally available unordered query evaluation* methods with CQE. We showed that an unordered query evaluation is maximally available only if, for every input instance, there exists an appropriate ordering of all possible queries, such that the given unordered query evaluation provides identical answers with a CQE that is based on the ordering.

In the results of the previous section, concerning maximal availability, we focused on the policy method of *known potential secrets*. We believe, however, that the same techniques should allow us to prove corresponding results for all four combinations of policy methods and user awareness, i.e., unknown potential secrets and both known and unknown secrecies, and for all enforcement policies for which existing methods of controlled query evaluation are applicable.

References

1. Bao, J., Slutzki, G., and Honavar, V., Privacy-Preserving Reasoning on the Semantic Web, 2007 IEEE/WIC/ACM International Conference on Web Intelligence, WI 2007, pp. 791-797

2. Biskup, J., For Unknown Secrecies Refusal is Better Than Lying, Data and Knowledge Engineering, Vol. 33 (2000), pp. 1-23
3. Biskup, J., and Bonatti, P.A., Lying Versus Refusal for Known Potential Secrets, Data and Knowledge Engineering, Vol. 38 (2001), pp. 199-222
4. Biskup, J., and Bonatti, P.A., Confidentiality Policies and their Enforcement for Controlled Query Evaluation, Proceedings of the 7th European Symposium on Research in Computer Security, ESORICS 2002, Lecture Notes in Computer Science, Vol. 2502, pp. 39-54
5. Biskup, J., and Bonatti, P., Controlled Query Evaluation for Enforcing Confidentiality in Complete Information Systems, International Journal of Information Security, Vol. 3 (2004), pp. 14-27
6. Biskup, J., and Bonatti, P.A., Controlled Query Evaluation for Known Policies by Combining Lying and Refusal, Annals of Mathematics and Artificial Intelligence, Vol. 40 (2004), pp. 37-62
7. Biskup, J., Burgard, D.M., Weibert, T., and Wiese, L., Inference Control in Logic Databases as a Constraint Satisfaction Problem, Proceedings of the 3rd International Conference on Information Systems Security, ICISS 2007, pp. 128-142
8. Biskup, J., and Weibert, T., Keeping Secrets in Incomplete Databases, International Journal of Information Security, Vol. 7 (2008), pp. 199-217
9. Bonatti, P.A., Kraus, S., and Subrahmanian, V.S., Foundations of Secure Deductive Databases, IEEE Transactions of Knowledge and Data Engineering, Vol. 7, No. 3 (1995), pp. 406-422
10. Biskup, J., and Wiese, L., On Finding an Inference-Proof Complete Database for Controlled Query Evaluation, Proceedings of the 20th Annual IFIP WG 11.3 Working Conference on Data and Applications Security, Data and Applications Security XX (2006), pp. 30-43
11. Chang, L.W., and Moskowitz, I.S., A Study of Inference Problems in Distributed Databases, Proceedings of the 16th Annual IFIP WG 11.3 Conference on Data and Applications Security, Data and Applications Security XVI (2002), pp. 191-204
12. Farkas, C., and Jajodia, S., The Inference Problem: A Survey, ACM SIGKDD Explorations Newsletter, Vol. 4, No. 2 (2002), pp. 6-11
13. Hale, J., and Shenoi, S., Analyzing FD Inference in Relational Databases, Data and Knowledge Engineering, Vol. 18 (1996), pp. 167-183
14. Sicherman, G.L., de Jonge, W., and van de Riet, R.P., Answering Queries Without Revealing Secrets, ACM Transactions on Database Systems, Vol. 8, No. 1 (1983), pp. 41-59
15. Stoffel, K., and Studer, T., Provable Data Privacy, Database and Expert Systems Applications, DEXA 2005, pp. 324-332
16. Stouppa, P., and Studer, T., A Formal Model of Data Privacy, 6th International Andrei Ershov Memorial Conference, Perspectives of Systems Informatics, PSI 2006, pp. 400-408
17. Stouppa, P., and Studer, T., Data Privacy for ALC Knowledge Bases, Logical Foundations of Computer Science, LFCS 2009, pp. 409-421
18. Voutsadakis, G., Slutzki, G., and Honavar, V., Secrecy Preserving Reasoning Over Entailment Systems Theory and Applications, Technical Report, Department of Computer Science, Iowa State University
19. Yang, X., and Li, C., Secure XML Publishing Without Information Leakage in the Presence of Data Inference, Proceedings of the 30th Very Large Data Base Conference, VLDB 2004, pp. 96-107

RECENT RESEARCH REPORTS

#126 Thomas Vestskov Terney. *The Combined Usage of Ontologies and Corpus Statistics in Information Retrieval*. PhD thesis, Roskilde, Denmark, August 2009.

#125 Jan Midtgaard and David Van Horn. Subcubic control flow analysis algorithms. 32 pp. May 2009, Roskilde University, Roskilde, Denmark.

#124 Torben Braüner. Hybrid logic and its proof-theory. 318 pp. March 2009, Roskilde University, Roskilde, Denmark.

#123 Magnus Nilsson. *Arbejdet i hjemmeplejen: Et etnometodologisk studie af IT-støttet samarbejde i den københavnske hjemmepleje*. PhD thesis, Roskilde, Denmark, August 2008.

#122 Jørgen Villadsen and Henning Christiansen, editors. *Proceedings of the 5th International Workshop on Constraints and Language Processing (CSLP 2008)*, Roskilde, Denmark, May 2008.

#121 Ben Schouten and Niels Christian Juul, editors. *Proceedings of the First European Workshop on Biometrics and Identity Management (BIOID 2008)*, Roskilde, Denmark, April 2008.

#120 Peter Danholt. *Interacting Bodies: Posthuman Enactments of the Problem of Diabetes Relating Science, Technology and Society-studies, User-Centered Design and Diabetes Practices*. PhD thesis, Roskilde, Denmark, February 2008.

#119 Alexandre Alapetite. *On speech recognition during anaesthesia*. PhD thesis, Roskilde, Denmark, November 2007.

#118 Paolo Bouquet, editor. *CONTEXT'07 Doctoral Consortium Proceedings*, Roskilde, Denmark, October 2007.

#117 Kim S. Henriksen. *A Logic Programming Based Approach to Applying Abstract Interpretation to Embedded Software*. PhD thesis, Roskilde, Denmark, October 2007.

#116 Marco Baroni, Alessandro Lenci, and Magnus Sahlgren, editors. *Proceedings of the 2007 Workshop on Contextual Information in Semantic Space Models: Beyond Words and Documents*, Roskilde, Denmark, August 2007.

#115 Paolo Bouquet, Jérôme Euzenat, Chiara Ghidini, Deborah L. McGuinness, Valeria de Paiva, Luciano Serafini, Pavel Shvaiko, and Holger Wache, editors. *Proceedings of the 2007 workshop on Contexts and Ontologies Representation and Reasoning (C&O:RR-2007)*, Roskilde, Denmark, August 2007.