A Top-Down Proof Procedure for Generalized Data Dependencies

Stéphane Coulondre

LISI, Bât Blaise Pascal (501), INSA / University of Lyon
20 Av Albert Einstein, 69621 Villeurbanne Cedex, France
Stephane.Coulondre@insa-lyon.fr

Abstract. Data dependencies are well known in the context of relational database. They aim to specify constraints that the data must satisfy to model correctly the part of the world under consideration. The implication problem for dependencies is to decide whether a given dependency is logically implied by a given set of dependencies. A proof procedure for the implication problem, called "chase", has already been studied in the generalized case of tuple-generating and equality-generating dependencies. The chase is a bottom-up procedure: from hypotheses to conclusion, and thus is not goal-directed. It also entails in the case of TGDs the dynamic creation of new constants, which can turn out to be a costly operation. This paper introduces a new proof procedure which is top-down: from conclusion to hypothesis, that is goal-directed. The originality of this procedure is that it does not act as classical theorem proving procedures, which require a special form of expressions, such as clausal form, obtained after Skolemization. We show, with our procedure, that this step is useless, and that the notion of *piece* allows infering directly on dependencies, thus saving the cost of Skolemizing the dependencies set and, morever, that the inference can be performed without dynamically creating new constants. Although top-down approaches are known to be less efficient in time than bottom-up ones, the notion of piece cuts down irrelevant goals usually generated, leading to a usable top-down method. With the more recent introduction of constrained and ordered dependencies, some interesting perspectives also arise.

Dependencies, Chase, Implication problem, TGD, EGD, Top-down

1 Introduction

Dependency theory allows the expression and modelling of constraints that the data must satisfy in order to reflect correctly the world that a relational database intends to describe. Since the introduction of functional dependencies by Codd ([Cod72]), many kinds of dependencies have been studied in the litterature, and a lot of work has been carried out in the late 70's and early 80's. Database dependencies theory is still an active area of research [SF00] [Her95] [LL97a] [LL97b] [LL98]. Functional and multivalued dependencies are the most known

classes of data dependencies. In practice, these two kinds are sufficient in general to express constraints ([Ull88]). Nevertheless, more general classes have been introduced, with the purpose of finding a uniform way to express constraints ([BV81],[SU82]). This paper deals with the class commonly known for generalizing most of the dependencies: that of tuple-generating and equality-generating dependencies (TGDs and EGDs) ([BV84b]). For a survey on this general class of dependencies, we refer the reader to [LL99] or [FV86].

The central problem is the implication problem, which is to decide whether a dependency is logically implied by a given set of dependencies. A process that could solve this problem would provide a solution to find the minimal cover of a set of dependencies, to decide whether a dependency is redundant within a set, useful during the constraint acquisition stage, etc. A procedure has already been designed in [BV84b] for that purpose: the well-known *chase*. Unfortunately, as the implication problem for TGDs and EGDs is semi-decidable, the *chase* is only a proof procedure, and therefore the process may run forever. As we argue in this paper, the *chase* is clearly a bottom-up procedure: from hypotheses to conclusion. Also, the *chase* entails the dynamic creation of new constants in the general case of TGDs.

We introduce a new proof procedure which is top-down: from conclusion to hypothesis that is goal-directed. The originality of this procedure is that it does not act as classical theorem proving procedures, by requiring a special form of the dependencies set, such as clausal form, obtained after Skolemization. We show, with our procedure, that this step is useless, and that the notion of *piece* allows infering directly on the dependencies set.

Therefore, our top-down chase is not simply the usual chase reversed, but a new way of solving the implication problem. The fact it can be performed top-down is the first contribution of this paper. The second contribution is to avoid dynamic creation of constants, as well as Skolemization of the dependencies set, usually applied on the original knowledge base prior to top-down proofs. This is realized by taking advantage of the form of the dependencies. To our knowledge, this has not been realized before. Indeed, dynamic creation of constants can be costly in proof procedures, such as the *chase*, in order to take into account the effect of existential quantifiers.

While it is true that top-down approaches can take exponential time longer w.r.t. bottom-up approaches, many reasons allow us to think that the proof procedure presented in this paper is efficient. These arguments will be detailed in the last section. The efficiency w.r.t. the bottom-up chase is currently being assessed in details. ???

More recently, constrained dependencies ([Mah94], [BCW95], [Mah97]) and ordered dependencies [Ng99] have been introduced. They originate in the constraint programming field and permit expression of semantic relations on variables, thus giving them an interpretation. The chase procedure has been redesigned in [MS96], still in a bottom-up way, in order to deal with constrained tuple-generating dependencies. This work in the dependency theory gives new perspectives for the top-down chase procedure we present.

The top-down chase originates in the conceptual graphs model, which is a knowledge representation model introduced by Sowa ([Sow84]). The base model has been extended with graph rules and an inference method, called *piece resolution* ([SM96]). The logical roots of this process have been studied in [CS98], and constitute the basis of the top-down chase. Proofs of the lemmas and theorems of this paper are therefore derived from these two last-mentioned.

Section 2 describes the framework and the implication problem for data dependencies. We sketch the existing (bottom-up) chase. In section 3 the top-down chase is explained. Section 4 closes the paper with some concluding remarks.

2 The Framework

The following definitions are for the most part taken from [BV84b], though simplified for the purpose of this paper. The first subsection states some assumptions we make throughout this paper. The second subsection presents the necessary definitions. The third subsection is a description of the kind of dependencies this paper focus on. Note that they are known to capture the semantics of most of the dependency types studied in the litterature. The fourth subsection presents formally the implication problem and then describes the traditional chase procedure, which has been designed to solve the implication problem. The chase is clearly a bottom-up mechanism, from hypotheses to conclusion.

2.1 Preliminaries

For sake of clarity, we assume several restrictions on the model. First, we assume that the universal relation assumption holds, i.e. the database is modelled with only one relation, because it usually permits a simpler formal presentation of the approaches. Secondly, dependencies are typed (many-sorted), so attribute domains are disjoint. Thirdly, we do not address dependencies with constants, as the last one illustrated below.

TGDs and EGDs can all be expressed in first-order logic. Informally speaking, an *equality-generating dependency* (EGD) says that if some tuples exist in the database, then some values in these tuples must be equal. A *tuple-generating dependency* (TGD) says that if some tuples exist in the database, then some other tuples, whose values are not necessarily taken from the first tuples, must also exist in the database. Thus some values may be unknown.

For example, to express a classical functional dependency stating that two customers having the same name also have the same identifier, we use the following EGD which is also a functional dependency :

$$\forall custid_1\ \forall custid_2\ \forall custname\ (cust(custid_1, custname) \wedge cust(custid_2, custname) \rightarrow custid_1 = custid_2)$$

To express that a invoice is always related to an existing order, we use the following TGD :

$$\forall invoiceid\ \forall amount\ \forall orderid\ (invoice(invoiceid,\ amount,\ orderid) \rightarrow \exists custid\ order(custid,\ orderid))$$

By introducing constants, we can state some more specialized constraints. For example, the invoice 23 is related to an order taken by the customer 12 :

$$\forall amount\ \forall orderid\ (invoice(23,\ amount,\ orderid) \rightarrow order(12,\ orderid))$$

However, the framework used in this paper is that of tableaux, because it is well-suited when the universal relation assumption holds. On the other hand, this assumption is very unpractival for real-world applications, because it would imply modelling the whole database schema with only one relation. That is the reason why we do not use real-world examples throughout this paper. Note that these restrictions appear here for clarity reasons only and all the results are applicable in the unrestricted model.

2.2 Attributes, relations and tableaux

Definition 1 (Attribute). *Attributes are symbols taken from a given finite set* $U = \{A_1, \ldots, A_n\}$ *called the* universe. *All sets of attributes are subsets of* U. *The complement of an attribute* A *in* U *is denoted by* $\overline{A}$.

Definition 2 (Relation). *With each attribute* A *is associated an infinite set, its* domain, *denoted* $DOM(A)$, *such that* $DOM(A) \cap DOM(B) = \emptyset$ *for* $A \neq B$. *Let* $Dom = \bigcup_{A \in U} DOM(A)$. *For an attribute set* X, *an* X*-value is a mapping* $w : X \rightarrow Dom$, *such that* $w(A) \in DOM(A)$ *for all* $A \in X$. *A* tuple *is a* U*-value. A* relation *is a set (not necessarily finite) of tuples. We use the letters* $t, u, v, w, \ldots$ *to denote tuples, and* $I, J, \ldots$ *to denote relations. For a tuple* w *and a set* $Y \subseteq U$, *we denote the restriction of* w *to* Y *by* $w[Y]$. *For an attribute* A, *we do not distinguish between* $w[A]$ *(which is an* A*-value) and* $w(A)$ *(which is an element of* $DOM(A)$*). Let* I *be a relation. The set of* X*-values in* I *is* $I[X] = \{w[X] | w \in I\}$. *The set of values in* I *is* $VAL(I) = \bigcup_{A \in U} I[A]$.

Definition 3 (Valuation). *A valuation is a mapping* $h : Dom \rightarrow Dom$, *such that* $a \in DOM(A)$ *implies* $h(a) \in DOM(A)$ *for all* $a \in Dom$. *The valuation* h *can be extended to tuples and relations as follows. Let* w *be a tuple. Then* $h(w) = h \circ w$ *(where* $\circ$ *denotes functional composition). Let* I *be a relation. Then* $h(I) = \{h(w) | w \in I\}$. *Usually, we are interested only in a small subset of* Dom, *for example,* $VAL(I)$. *We let* h *be undefined for other values, and say that* h *is a valuation on* I. *We associate with each attribute* A *an infinite set of constants* $CONS(A) \subseteq DOM(A)$. *Let* $CONS = \bigcup_{A \in U} CONS(A)$. *For any valuation* h *and a constant* $a \in CONS(A)$, *we require* $h(a) = a$. *Let* I *be a relation,* $CONS(I)$ *is the set of constants appearing in* $VAL(I)$.

Example 1. Let I be a relation, and w be a tuple, as shown below. Let h be a valuation on w such that $a_1 \mapsto a_1$, $b_1 \mapsto b_1$, $c_1 \mapsto c_1$ and $d_1 \mapsto d_2$. Then $h(w) \in h(I)$.

I : w :

A	B	C	D
a_1	b_1	c_1	d_2
a_1	b_2	c_2	d_1

A	B	C	D
a_1	b_1	c_1	d_1

▲

Definition 4 (Extension). *Let I be a relation, and let h be a valuation on I. An* extension *of h to another relation I' is a valuation h' on I', which agrees with h on $VAL(I)$. If h is the identity relation on $VAL(I)$, then we say that h is the identity on I.*

2.3 Dependencies

An *equality-generating dependency* (EGD) says that if some tuples exist in the database, then some values in these tuples must be equal.

Definition 5 (EGD). *An Equality-Generating Dependency (EGD) is a finite pair $< (a_1, a_2), I >$, where a_1 and a_2 are A-values for some attribute A, and I is a finite relation, such that $a_1, a_2 \in I[A]$. We also call this EGD an A-EGD. A relation J satisfies $< (a_1, a_2), I >$ if, for any valuation h such that $h(I) \subset J$, we have $h(a_1) = h(a_2)$. Note that, if $a_1 = a_2$, then $< (a_1, a_2), I >$ is trivially satisfied by every relation and the EGD is trivial.*

Example 2. Consider the relation I of example 1. Then $E =< (c_1, c_2), I >$ is an EGD, as shown below, and K satisfies E. Note that E is equivalent to the following functional dependency : $A \rightarrow C$.

$< (c_1, c_2), I >$: K :

A	B	C	D
$c_1 = c_2$			
a_1	b_1	c_1	d_2
a_1	b_2	c_2	d_1

A	B	C	D
a'	b	c	d
a'	b	c	d'
a	b	c'	d

▲

A *tuple-generating dependency* (TGD) says that if some tuples exist in the database, then some other tuples, whose values are not necessarily taken from the first tuples, must also exist in the database. Thus some values may be unknown.

Definition 6 (TGD). *A Tuple-Generating Dependency (TGD) is a pair of finite relations $< I', I >$. It is satisfied by a relation J if for any valuation h, such that $h(I) \subset J$, there is an extension h' of h to I' so that $h'(I') \subset J$.*

Example 3. Consider the relation I of example 1. Let I' be another relation, as shown below. Then $T =< I', I >$ is a TGD and K' satisfies T.

I' : $\quad T =< I', I >$:

A	B	C	D
a_2	b_1	c_2	d_2
a_2	b_2	c_1	d_2

A	B	C	D
a_2	b_1	c_2	d_2
a_2	b_2	c_1	d_2
a_1	b_1	c_1	d_2
a_1	b_2	c_2	d_1

K' :

A	B	C	D
a	b	c	d
a	b'	c'	d
a	b'	c	d
a	b	c'	d

▲

2.4 The implication problem and the (bottom-up) chase

Let D be a set of dependencies, and d be a dependency. The implication problem is to decide whether $D \models d$, that is to determine whether d is true in every database in which each dependency of D is true. Let $SAT(D)$ be the set of all relations, only composed of constants, that satisfy all the dependencies in D, then the implication problem is equivalent to decide whether $SAT(D) \subseteq SAT(d)$.

The implication problem can also be considered under a different view: that of finite implication problem. The finite implication problem is to decide whether d is satisfied by every finite relation that satisfies all dependencies in D. However, as detailed below, this problem admits no proof procedure.

The chase procedure has been designed to solve the implication problem. The reader can refer to [BV84b] for a complete description. *Intuitively speaking*, if the dependency to be proven is the TGD $< I', I >$, or the EGD $< (a_1, a_2), I >$, the chase procedure takes the relation I, and treats it as if it formed a set of tuples. Then it applies repeatedly the dependencies of D, following two distinct rules, one for TGDs, whose effect is to add tuples to the relation, and one for EGDs, whose effect is to "identify" two constants. When the dependency to be proven is a TGD, the procedure stops when obtaining the tuples of I'. When it is a EGD, the procedure stops when obtaining the identification of a_1 and a_2.

This mechanism has been shown to be sound and complete in [Var84]. Note that the implication problem for TGDs and EGDs is semi-decidable. Thus the chase may not stop.

The chase procedure is clearly a bottom-up (or forward chaining) one. Indeed, rule applications are generating new tuples or identification of constants. This is executed until we obtain the desired conclusion. The goal to be proven is not used to guide the process. Moreover, when applying a TGD rule whose effect is

to add tuples to the relation, existential quantification always requires a costly dynamic creation of new constants.

We now show that we can apply a top-down (or backward) procedure, in which the process is goal-directed.

3 The Top-Down Chase

Depending on the type of the dependencies, the implication problem is solvable or recursively unsolvable ([BV81,Var84,GL82]). This means that in the first case, there is a decision procedure, hence an algorithm that always halt, whereas in the second case, there is a proof procedure: if the implication is true, then the process will terminate. In the other case, it might never stop. The finite implication problem, on the other hand, is not even partially solvable. Therefore, there can be no proof procedure for it.

A subset of TGDs is known as Full TGDs. These dependencies have the same form as Datalog rules. In this case, the notion of *piece*, stated in the introduction, is useless. Therefore, dedicated top-down theorem proving procedures can be applied to solve the implication problem involving only Full TGDs, such as Query-Subquery ([Vie86]) and OLD-Resolution ([TS86]). The principle aim of these procedures is to provide a terminating algorithm for top-down evaluation of Datalog rules. Indeed, even if the implication problem is decidable for this particular class, their top-down evaluation might not terminate. To tackle this problem, tabulation (or memoing) techniques must be applied in order to cut down looping sequences. Notice that these techniques do not affect completeness. In the general case of TGDs, however, there are unfinishing sequences that remain undetectable, due to the undecidable nature of the implication problem. In the case of Full TGDs, the implication problem and the finite implication problem coincide and are both decidable.

For this kind of dependencies, as well as all other subsets (functional and multivalued dependencies) the interest of the top-down chase is to be also applicable. However, in these particular cases, specific proof procedures shall be more efficient because they implement memoing. The top-down chase and the classical bottom-up chase are clearly needed when dependencies are more complex (i.e. not a decidable subset), and also to provide a general way to solve the implication problem. Notice that memoing can be added to the top-down chase too, but this is an implementation-level extension.

There have been a lot of work carried out also in the active database area. For a survey, we refer the reader to [WC95]. Active databases generally support the automatic triggering of updates in response to internal or external events. These updates are expressed by mean of rules which are slightly different from TGDs and EGDs. Usually, these rules can be expressed in Datalog with negation [BDR98] [AHV95]. The main difference lies in the fact that the variables occuring in the conclusion are all universally quantified, whereas in database dependencies they are existentially quantified if they do not occur in the hypothesis too.

The first subsection presents a proof procedure for TGDs only. The second subsection shows how this procedure can take EGDs into account as well, by using some reduction theorems.

3.1 A proof procedure for TGDs

Let D be a set of TGDs without constants, and let $d =< J', J >$ be a TGD without constants. *Intuitively speaking*, to decide whether $D \models d$, we start with the tuples of J' and treat it as if it formed a relation. Let Q be this relation. Q is considered as the *goal* to reach. On the other hand, we add J to D, after replacing each symbol of J by a new constant, by transforming each tuple into a TGD with constants. Then we try to remove the tuples of Q, by applying successively a rule, giving each time a new goal. These rule applications may introduce new tuples. If we achieve in removing all tuples, i.e. obtaining an empty goal, then $D \models d$.

Compared with classical theorem proving methods, this proof procedure relies on a complex core rule that does not require any modification of the dependencies in D. Indeed, in order to apply the classical resolution method, one needs to rewrite D and $\neg d$ under clausal form. In order to do so, the Skolemization step would generate constants or functions due to the presence of existential quantifiers in the conclusion. Then the resolution method would try to generate the empty clause. When obtained, then the Herbrand theorem ensures that $D \models d$.

In the present case, the top-down proof procedure does not require rewriting the dependencies of D nor dynamically creating new constants or functions. Indeed, the originality is to allow inferring directly on dependencies of D, thus providing a general mechanism. Notice that this can be obtained thanks to the particular form of TGDs and EGDs.

Let us now explain this process in a more formal way. We need the notion of *distinct substitution* that replaces each symbol of a relation by a new constant.

Definition 7 (Distinct substitution). *Let J be a relation without constants. A* distinct substitution *ω on J is a valuation on J such that $\omega(a)$ is a new distinct constant for each $a \in VAL(J)$.*

Now, we can add to D a set of TGDs with constants, each of them being of the form $< u, \emptyset >$, with u corresponding to a distinct tuple in $\omega(J)$. Thus we add $|\omega(J)|$ TGDs. The following theorem states that checking whether $D \models d$ can be reduced to checking whether $D_\omega \models Q$, where D_ω is the result of adding the new TGDs to D, and Q is the *goal*. Note that the added TGDs are the only ones of D_ω having constants in it. More formally, we have :

Theorem 1. *Let D be a set of TGDs without constants, and let $d =< J', J >$ be a TGD without constants. Let ω be a distinct substitution on J. Let ω' be an extension of ω on J' such that ω' is the identity on $VAL(J') - VAL(J)$. Let D_ω denote $D \cup \{< u, \emptyset > | u \in \omega(J)\}$ and let Q denote the TGD $< \omega'(J'), \emptyset >$ with possibly some constants. Then $D \models d$ if and only if $D_\omega \models Q$. Q is also called a* goal.

Example 4. Let d be a TGD without constants, as shown below.

$$d =< J', J > :$$

A	B	C	D
a_3	b_4	c_4	d_5
a_4	b_4	c_3	d_5
a_4	b_3	c_4	d_5
a_3	b_3	c_3	d_3
a_3	b_4	c_4	d_4

Let ω be a distinct substitution on J such that $a_4 \mapsto a$, $a_5 \mapsto a'$, $b_3 \mapsto b$, $b_4 \mapsto b'$, $c_3 \mapsto c$, $c_4 \mapsto c'$ and $d_3 \mapsto d$, with a, a', b, b', c, c', d being constants. Let ω' be an extension of ω on J' such that ω' is the identity on $VAL(J') - VAL(J)$. Let D_ω denote $D \cup \{< u, \emptyset > | u \in \omega(J)\}$:

$$\{< u, \emptyset > | u \in \omega(J)\} :$$

A	B	C	D
a	b	c	d

A	B	C	D
a	b'	c'	d'

Let Q denote the TGD $< \omega'(J'), \emptyset >$:

$$Q =< \omega'(J'), \emptyset > :$$

A	B	C	D
a	b'	c'	d_5
a_4	b'	c	d_5
a_4	b	c'	d_5

Then $D \models d$ if and only if $D_\omega \models Q$.

▲

We now describe the main step of the process. Given a TGD and a goal, this rule constructs a new goal. We need to introduce the notion of piece. A piece is a set of tuples that are semantically linked. When applying the rule, we will see that these tuples can be treated at the same time. This notion is an alternative to the classical Skolemization process that replaces unknown values by constants, prior to using classical first-order logic provers. A piece is a set of tuples which share unknown values, and are therefore treated as a whole. This is the originality of the top-down chase.

Definition 8 (Piece). *Let L be a relation, and $v \subseteq VAL(L)$. Pieces of L in relation to v are defined in the following way : for all tuples $u, v \in L$, u and v belong to the same piece if and only if there is a sequence of tuples $(u_1, \ldots, u_m)$ of L such that $u_1 = u$ and $u_m = v$ and $\forall i = 1 \ldots m-1$, $\exists a \in v$ such that*

$a \in VAL(u_i)$ *and* $a \in VAL(u_{i+1})$. *As a corollary, if a tuple does not share any element of* v *with another, it forms itself a piece. By construction, the set of pieces is unique for a given relation* L *and a given set of symbols* v.

Example 5. Let L be a relation, as shown below, and $v = \{a_2, c_1, d_2\}$. Then there are three pieces of L in relation to v : P_1 is containing tuples of L that are sharing the symbols a_2 and c_1, P_2 is containing tuples of L that are sharing the symbol d_2, and P_3 is containing the last tuple of L that does not share any of these symbols and therefore is itself a piece.

L :

A	B	C	D
a_1	b_3	c_1	d_1
a_2	b_4	c_3	d_3
a_2	b_1	c_1	d_1
a_1	b_2	c_2	d_2
a_1	b_3	c_3	d_2
a_1	b_1	c_3	d_1

P_1 :

A	B	C	D
a_1	b_3	c_1	d_1
a_2	b_4	c_3	d_3
a_2	b_1	c_1	d_1

P_2 :

A	B	C	D
a_1	b_2	c_2	d_2
a_1	b_3	c_3	d_2

▲

We now define the core rule.

Definition 9 (core rule). *Let* $T =< I', I >$ *be a TGD of* D_ω *and* $Q_n =< V, \emptyset >$ *be a goal, such that* $(VAL(V) \cap (VAL(I) \cup VAL(I')))\backslash CONS = \emptyset$. *If there is a valuation* h *on* $I \cup I' \cup V$ *such that:*

- $\forall a \in VAL(I) - VAL(I')$, $h(a) = a$
- $\forall a \in VAL(I') - VAL(I)$, $h(a) = a$
- there is a piece P of $h(V)$ in relation to $(VAL(I') - VAL(I))\backslash CONS$, such that $P \subseteq h(I')$

then the result of the application of the rule, denoted $Q_{n+1} = \mathcal{T}(T, Q_n)$, is $Q_{n+1} =< h(V)\backslash P \cup h(I), \emptyset >$. Q_{n+1} becomes a new goal. Thus it is obtained by removing P from $h(V)$ and by adding $h(I)$. Note that there may be several possible TGDs obtained by an application of the rule, depending on the valuation h. In a sense, the piece notion allows us to group together some tuples according to some particular symbols (which correspond in logic to existentially quantified variables of T) they share.

Example 6. Consider the TGD T of example 3, and the TGD Q of example 4, here called Q_n, as shown below. Let $V = \omega'(J')$. Therefore, $Q =< V, \emptyset >$. We can check that $(VAL(V) \cap (VAL(I) \cup VAL(I')))\backslash CONS = \emptyset$. Let h be a valuation on $I \cup I' \cup V$ such that $\forall a \in VAL(I) - VAL(I')$, $h(a) = a$ and $\forall a \in VAL(I') - VAL(I)$, $h(a) = a$. Therefore $h(a_2) = a_2$. h is also defined by: $h(a_4) = a_2$, $h(b_1) = b'$, $h(b_2) = b$, $h(c_1) = c'$, $h(c_2) = c$, and $h(d_5) = d_2$.

$$T =< I', I > : Q_n =< V, \emptyset >:$$

A	B	C	D
a_2	b_1	c_2	d_2
a_2	b_2	c_1	d_2
a_1	b_1	c_1	d_2
a_1	b_2	c_2	d_1

A	B	C	D
a	b'	c'	d_5
a_4	b'	c	d_5
a_4	b	c'	d_5

$$T' =< h(I'), h(I) > : Q'_n =< h(V), \emptyset >:$$

A	B	C	D
a_2	b'	c	d_2
a_2	b	c'	d_2
a_1	b'	c'	d_2
a_1	b	c	d_1

A	B	C	D
a	b'	c'	d_2
a_2	b'	c	d_2
a_2	b	c'	d_2

There are two pieces of $h(V)$ in relation to $(VAL(I') - VAL(I))\backslash CONS = \{a_2\}$, which are the first tuple of $h(V)$ and the last two tuples of $h(V)$. The second piece P is such that $P \subseteq h(I')$. We construct $Q_{n+1} = \mathcal{T}(T, Q_n)$. Let $V' = h(V)\backslash P \cup h(I)$:

$$Q_{n+1} =< V', \emptyset >:$$

A	B	C	D
a	b'	c'	d_2
a_1	b'	c'	d_2
a_1	b	c	d_1

▲

The following theorem states that a goal is implied by a set of TGDs if and only if there is a sequence of rule applications such that the sequence starts with the original goal, and gives the empty goal as an end result. The proof is detailed at the end of the paper.

Theorem 2 (Soundness and Completeness). *$D_\omega \models Q$ if and only if there is a sequence $((T_1, Q_1), \ldots, (T_n, Q_n))$ of rule applications, such that $T_1, \ldots T_n \in D_\omega$ and $Q_1 = Q$ and $Q_n =< \emptyset, \emptyset >$ and $\forall i \in [1..n-1]$, $Q_{i+1} = \mathcal{T}(T_i, Q_i)$. Note that the last TGD used, T_n, is of the form $< u, \emptyset >$.*

It is important to notice that whenever $D_\omega \models Q$, theorem 2 ensures that there is at least one sequence of rule applications, but it does not give a method to find it. Therefore, to implement the top-down chase, we need to add a search strategy (breadth-first or depth-first for example). In order to illustrate the need for a search strategy, let us detail in the following three kinds of sequences for the same set of TGDs D_ω and the same goal Q. The first one stops and ends with an empty goal, thus proving the implication. The second one does not terminate (although in this particular case, the loop might be detected and stopped), and the third one is stuck (no rule can be applied any more) and thus needs backtracking whenever the search algorithm is depth-first.

In the following examples, we will consider that D_ω contains the TGD T of example 6, as well as another TGD S detailed below. Remember also that in 4 it has been shown that in order to prove that a dependency d is implied by a set of dependencies D, we construct a goal to be proven (i.e. without hypothesis), and we add some TGDs (also without hypothesis) to D, giving D_ω. Therefore, in the present case, D_ω contains T as well as two more TGDs presented in example 6. Finally, the goal to be proven is Q, already presented in example 4 . D_ω is detailed below:

U_1 : U_2 :

A	B	C	D
a	b	c	d

A	B	C	D
a	b'	c'	d'

T : S :

A	B	C	D
a_2	b_1	c_2	d_2
a_2	b_2	c_1	d_2
a_1	b_1	c_1	d_2
a_1	b_2	c_2	d_1

A	B	C	D
a_1	b_2	c_1	d_2
a_1	b_1	c_2	d_3
a_1	b_1	c_1	d_1
a_1	b_2	c_2	d_1

And Q is :

Q:

A	B	C	D
a	b'	c'	d_5
a_4	b'	c	d_5
a_4	b	c'	d_5

Example 7. The following example illustrates a successful proof sequence of rule applications, ending with the empty goal. Consider the first rule application of example 6, giving a new goal $Q_1 = \mathcal{T}(T, Q)$ to be proven :

Q_1 :

A	B	C	D
a	b'	c'	d_2
a_1	b'	c'	d_2
a_1	b	c	d_1

Consider the TGD U_1. Let $U_1 =< I', I >$ and $Q_1 =< V_1, \emptyset >$. We can check that $(VAL(V_1) \cap (VAL(I) \cup VAL(I')))\backslash CONS = \emptyset$. Let h be a valuation on $I \cup I' \cup V_1$ such that $\forall a \in VAL(I) - VAL(I')$, $h(a) = a$ and $\forall a \in VAL(I') - VAL(I)$, $h(a) = a$ (trivial). h is also defined by: $h(a_1) = a$ and $h(d_1) = d$. Therefore we have :

$$Q'_1 =< h(V_1), \emptyset > :$$

A	B	C	D
a	b'	c'	d_2
a	b'	c'	d_2
a	b	c	d

As there is no element in $(VAL(I') - VAL(I))\backslash CONS$, each tuple of $h(V_1)$ is itself a piece. The third tuple P is such that $P \subseteq h(I')$. We construct $Q_2 = \mathcal{T}(U_1, Q_1)$. Let $Q_2 =< h(V_1)\backslash P \cup h(I), \emptyset > :$

Q_2 :

A	B	C	D
a	b'	c'	d_2
a	b'	c'	d_2

Now consider the TGD U_2. Let $U_2 =< I', I >$ and $Q_2 =< V_2, \emptyset >$. We can check that $(VAL(V_2) \cap (VAL(I) \cup VAL(I')))\backslash CONS = \emptyset$. Let h be a valuation on $I \cup I' \cup V_2$ such that $\forall a \in VAL(I) - VAL(I')$, $h(a) = a$ and $\forall a \in VAL(I') - VAL(I)$, $h(a) = a$ (trivial). h is also defined by: $h(d_2) = d'$. Therefore we have :

$$Q'_2 =< h(V_2), \emptyset > :$$

A	B	C	D
a	b'	c'	d'
a	b'	c'	d'

As there is no element in $(VAL(I') - VAL(I))\backslash CONS$, each tuple of $h(V_2)$ is itself a piece. The first tuple P is such that $P \subseteq h(I')$. We construct $Q_3 = \mathcal{T}(U_2, Q_2)$. Let $Q_3 =< h(V_2)\backslash P \cup h(I), \emptyset > :$

Q_3 :

A	B	C	D
a'	b'	c'	d'

Following the same schema, we apply the very same rule to Q_3, giving $Q_4 =< h(V_3)\backslash P \cup h(I), \emptyset >$:

Q_4 :

A	B	C	D

Therefore, we have proven that $D_\omega \models Q$, thus $D \models d$.

▲

Example 8. The following example illustrates a non-terminating sequence of rule applications. Consider the first rule application of example 6, giving the new goal $Q_1 = \mathcal{T}(T, Q)$ to be proven, as well as the TGD T :

Q_1 : T

A	B	C	D
a	b'	c'	d_2
a_1	b'	c'	d_2
a_1	b	c	d_1

A	B	C	D
a_2	b_1	c_2	d_2
a_2	b_2	c_1	d_2
a_1	b_1	c_1	d_2
a_1	b_2	c_2	d_1

Let $T =< I', I >$ and $Q_1 =< V_1, \emptyset >$. We can check that $(VAL(V_1) \cap (VAL(I) \cup VAL(I')))\backslash CONS = \{a_1, d_1, d_2\}$. Therefore, we must rename these symbols either in T or in Q_1. Let now Q_1 be :

Q_1 :

A	B	C	D
a	b'	c'	d_4
a_3	b'	c'	d_4
a_3	b	c	d_3

Let h be a valuation on $I \cup I' \cup V_1$ such that $\forall a \in VAL(I) - VAL(I')$, $h(a) = a$ and $\forall a \in VAL(I') - VAL(I)$, $h(a) = a$. Therefore $h(a_1) = a_1$, $h(a_2) = a_2$, and $h(d_1) = d_1$. h is also defined by: $h(a_3) = a_2$, $h(b_1) = b'$, $h(b_2) = b$, $h(c_1) = c$, $h(c_2) = c'$, $h(d_3) = d_2$ and $h(d_4) = d_2$. Therefore we have :

$$T' =< h(I'), h(I) > : Q'_1 =< h(V_1), \emptyset >:$$

A	B	C	D
a_2	b'	c'	d_2
a_2	b	c	d_2
a_1	b'	c	d_2
a_1	b	c'	d_1

A	B	C	D
a	b'	c'	d_2
a_2	b'	c'	d_2
a_2	b	c	d_2

There are two pieces of $h(V_1)$ in relation to $(VAL(I') - VAL(I))\backslash CONS = \{a_2\}$, which are the first tuple of $h(V_1)$ and the last two tuples of $h(V_1)$. The second piece P is such that $P \subseteq h(I')$. We construct $Q_2 = \mathcal{T}(T, Q_1)$. Let $V_2 = h(V_1)\backslash P \cup h(I)$:

$$Q_2 =< V_2, \emptyset >:$$

A	B	C	D
a	b'	c'	d_2
a_1	b'	c	d_2
a_1	b	c'	d_1

By applying rule T over Q_2 one more time, giving $Q_3 = \mathcal{T}(T, Q_2)$, we obtaain :

$$Q_3:$$

A	B	C	D
a	b'	c'	d_2
a_1	b'	c'	d_2
a_1	b	c	d_1

Therefore $Q_3 = Q_1$ w.r.t. symbol renaming, thus the sequence is entering a loop. Note that in this particular case, it is simple to detect and thus to stop. However, this is an implementation-level treatment.

▲

Example 9. The following example illustrates a stuck sequence (no rule can be applied any more) that thus needs backtracking whenever the search algorithm is depth-first. Consider again the goal $Q_1 = \mathcal{T}(T, Q)$ to be proven, as well as the TGD S :

Q_1 : S

A	B	C	D
a	b'	c'	d_2
a_1	b'	c'	d_2
a_1	b	c	d_1

A	B	C	D
a_1	b_2	c_1	d_2
a_1	b_1	c_2	d_3
a_1	b_1	c_1	d_1
a_1	b_2	c_2	d_1

Let $S =< I', I >$ and $Q_1 =< V_1, \emptyset >$. We can check that $(VAL(V_1) \cap (VAL(I) \cup VAL(I')))\backslash CONS = \{a_1, d_1, d_2\}$. Therefore, we must rename these symbols either in S or in Q_1. Let now Q_1 be :

Q_1 :

A	B	C	D
a	b'	c'	d_5
a_3	b'	c'	d_5
a_3	b	c	d_4

Let h be a valuation on $I \cup I' \cup V_1$ such that $\forall a \in VAL(I) - VAL(I'), h(a) = a$ and $\forall a \in VAL(I') - VAL(I)$, $h(a) = a$. Therefore $h(d_1) = d_1$, $h(d_2) = d_2$ and $h(d_3) = d_3$. h is also defined by: $h(a_1) = a$, $h(a_3) = a$, $h(b_1) = b$, $h(b_2) = b'$, $h(c_1) = c'$, $h(c_2) = c$, $h(d_4) = d_3$ and $h(d_5) = d_3$. Therefore we have :

$S' =< h(I'), h(I) > : Q_1' =< h(V_1), \emptyset >$:

A	B	C	D
a	b'	c'	d_2
a	b	c	d_3
a	b	c'	d_1
a	b'	c	d_1

A	B	C	D
a	b'	c'	d_2
a	b'	c'	d_2
a	b	c	d_3

There are two pieces P_1 and P_2 of $h(V_1)$ in relation to $(VAL(I') - VAL(I))\backslash CONS = \{d_2, d_3\}$, which are the first (identical) two tuples of $h(V_1)$ and the last tuple of $h(V_1)$. The two piece are included in $h(I')$. Note that we can save one sequence step by removing both pieces at the same time instead of performing two steps with the same valuation h. We construct $Q_2 = \mathcal{T}(T, Q_1)$. Let $V_2 = h(V_1)\backslash P_1 \backslash P_2 \cup h(I)$:

$Q_2 =< V_2, \emptyset >$:

A	B	C	D
a	b	c'	d_1
a	b'	c	d_1

One can now verify that there is no applicable rule to Q_2 amongst T, S, U_1 and U_2.
▲

3.2 Dealing with EGDs

EGDs add some difficulties because they include an equality predicate. Nevertheless, it has been shown in [BV84b] that the implication problem for TGDs and EGDs without constants is reducible to the implication problem for TGDs without constants. Thus, our core rule is sufficient.

Let e be the A-EGD $e =< (a_1, a_2), J >$. Let $w_1$be a tuple such that $w_1[A] = a_1$ and for all attributes $B \in \overline{A}$, $w_1[B] \notin J[B]$. Let w_2 be a tuple such that $w_2[\overline{A}] = w_1[\overline{A}]$ and $w_2[A] = a_2$. We associate with e two TGDs. e_1 is $< w_1, J \cup \{w_2\} >$, e_2 is $< w_2, J \cup \{w_1\} >$. For a given set D of TGDs and EGDs, let D^* be the result of replacing each EGD e in D by its two associated TGDs e_1 and e_2.

Example 10. Consider the EGD E of example 2. We associate to E two TGDs e_1 and e_2, as shown below.

w_1 : w_2 :

A	B	C	D
a_2	b_2	c_1	d_3

A	B	C	D
a_2	b_2	c_2	d_3

$e_1 =< w_1, I \cup \{w_2\} >$: $e_2 =< w_2, I \cup \{w_1\} >$:

A	B	C	D
a_2	b_2	c_1	d_3
a_1	b_1	c_1	d_2
a_1	b_1	c_2	d_1
a_2	b_2	c_2	d_3

A	B	C	D
a_2	b_2	c_2	d_3
a_1	b_1	c_1	d_2
a_1	b_1	c_2	d_1
a_2	b_2	c_1	d_3

▲

The following theorems allow us to ignore the presence of EGDs by reducing the implication of a dependency by a set of TGDs and EGDs to the implication of a TGD by a set of TGDs only:

Theorem 3. *([BV84b]) Let D be a set of TGDs and EGDs without constants and let d be a TGD without constants. Then $D \models d$ if and only if $D^* \models d$.*

Theorem 4. *([BV84b]) Let D be a set of TGDs and EGDs without constants. Let e be a non-trivial A-EGD without constants and let e_1 and e_2 be its two associated TGDs. Then $D \models e$ if and only if $D^* \models e_1$ and there is a non-trivial A-EGD in D.*

Thus, we can now generalize theorem 1 and give the final following theorem that reduces the implication problem of TGDs and EGDs to the existence of a top-down chase using the rule:

Theorem 5. *Let D be a set of TGDs and EGDs without constants:*

- Let $d =< I', I >$ be a TGD without constants. Let ω be a distinct substitution on I. Let D^*_ω denote $D^* \cup \{< u, \emptyset > | u \in \omega(I)\}$ and let Q denote the TGD $< \omega(I'), \emptyset >$. Then $D \models d$ if and only if $D^*_\omega \models Q$ if and only if there is a sequence $((T_1, Q_1), \ldots, (T_n, Q_n))$ of rule applications, such that $T_1, \ldots T_n \in D^*_\omega$ and $Q_1 = Q$ and $Q_n =< \emptyset, \emptyset >$ and $\forall i \in [1..n-1]$, $Q_{i+1} = \mathcal{T}(T_i, Q_i)$.
- Let e be a non-trivial A-EGD without constants and let $e_1 =< I', I >$ and e_2 be its two associated TGDs. Let ω be a distinct substitution on I. Let D^*_ω denote $D^* \cup \{< u, \emptyset > | u \in \omega(I)\}$ and let Q denote the TGD $< \omega(I'), \emptyset >$. Then $D \models e$ if and only if $D^*_\omega \models Q$ and there is a non-trivial A-EGD in D, if and only if there is a sequence $((T_1, Q_1), \ldots, (T_n, Q_n))$ of rule applications, such that $T_1, \ldots T_n \in D^*_\omega$ and $Q_1 = Q$ and $Q_n =< \emptyset, \emptyset >$ and $\forall i \in [1..n-1]$, $Q_{i+1} = \mathcal{T}(T_i, Q_i)$ and there is a non-trivial A-EGD in D.

4 Conclusion and remarks

As a conclusion, we discuss several points. First we compare the top-down chase with a backward formal system and with other proof procedures. Then we discuss the contributions of our work. We conclude this paper by lifting some restrictions on the model and pointing out some perspectives.

4.1 Comparison with formal systems

Many formal systems have been studied for data dependencies ([BFH77], [Sci82], [BV84a], [SU82]). In [BV84a], some formal systems for TGDs and EGDs are studied. Two of these systems are backward, but only one, namely the T3 formal system, has some similarities with the top-down chase. We sketch here the main differences. We refer the reader to this paper for more details about formal systems for TGDs and EGDs.

The T3 system deals with TGDs and their unknown values (i.e. non-constant symbols of $VAL(J') - VAL(J)$) in the following way : the process starts with a goal tuple Q and applies a TGD $T =< J', J >$ by making J' and J coincide. Typically this is achieved using especially the collapsing, augmentation and projection rules. When J' and J coincide, it uses the transitivity rule to derive a new goal. A derivation tree is not linear , as it is the case for SLD-resolution.

The top-down chase leads to a linear inference in the sense that it uses directly TGDs from the base without first applying rules on them. There is only one rule. Obviously, it is a more complex step.

4.2 Discussion

To our knowledge, this is the first time a top-down proof procedure is used to solve the implication problem for dependencies. As such, a first constribution is to have shown that this can be performed top-down.

The top-down chase avoids rewriting the dependencies set under clausal form, and avoids the corresponding Skolemization process, necessary in order to use classical top-down theorem proving procedures. In the top-down chase, only the preliminary transformation of the TGD to be proven requires creating as many new constants as universally quantified variables appearing in it. Thus this step is unsignificant. By providing a way to infer directly on the original form of dependencies, by way of the notion of piece, the top-down chase is conceptually speaking a simpler approach, this is our second contribution.

Compared to the classical chase, it does not entails the dynamic creation of new constants. This is our third contribution.

Note that the top-down chase is not the usual chase simply reversed. The core rule is totally different from those used in the bottom-up approach of [BV84b]. Indeed, it has been shown that there is a strong relationship between the bottom-up chase and resolution with paramodulation [BC90][NR01]. Whereas the latters acts on the Skolemized set of clauses, the top-down chase does not need this prior step.

We are currently investigating in details the efficiency of the top-down chase. Actually, this is not trivial to assess, because of the various parameters that come into play. We agree with [BMSU86] on the fact that " it is unreasonable to expect proofs of optimality, or even superiority of one algorithm over another, in all cases ".

However, we shall detail some of our arguments.

First of all, there is no need for Skolemizing the dependencies set nor dynamic constant creation. This might increase the efficiency. Indeed, on the contrary of the bottom-up chase, existentially quantified variables in the TGD conclusions never transform into constants. The only new symbols that are created are those of the current goal that are temporary variables. The goal can grow by the addition of hypotheses of the dependencies used by the $\mathcal{T}-$rule. Notice that these symbols are, on the other hand, deleted whenever they are mapped to existing constants, or whenever some pieces are removed from the goal. Depending on the search strategy, they are also deleted from memory whenever the sequence of rule applications is stuck or successful (and needs for example backtracking when doing depth-first search).

On the other hand, top-down approaches can take exponential time longer w.r.t. bottom-up approaches. However, the notion of piece performs some dynamic optimizations by dealing with groups of atoms instead of one at the time, thus failures are detected earlier w.r.t. classical top-down procedures, and irrelevant goals are cut down earlier. This feature dramatically reduces the number of backtracks. As an illustration, we have compared our method with Prolog over 5000 proofs generated at random, varying each time the size and the content of the dependencies set, and the size of the dependencies. The top-down

chase has been implemented on top of the CoGITo platform [GS98], which is a set of tools for conceptual graphs management. These tests showed that the top-down chase provides a logarithmic average improvement for the number of backtracks as well as for the proof duration (when the process stops), in spite of a non-optimized implementation. This reduces drastically the drawback of the top-down approach, and gives the top-down chase a fair *practical* level of efficiency. The fact that the search for the right symbol mappings has an exponential complexity does not practically seem to act as a brake. Indeed, whereas for SLD-Resolution, some irrelevant goals may lead to exploring a branch of the resolution tree, the top-down chase detects failures earlier and allows saving the average time.

However, either approach could run faster in practice on given data [BMSU86]. That is why we think presenting an example in which the top-down chase would be more efficient than the bottom-up chase would not be significant and would not provide any worthwhile addition to the discussion.

For all these reasons, we plan to implement the chase of [BV84b] in order to practically compare their efficiency over a random dataset. We think dependencies might be divided into classes for which one or the other approach would be better but identifying them is still an open problem.

We must mention that some optimisations to the bottom-up approach have been made by magic-sets in [MS96]. The principle of magic sets ([BMSU86]) is to perform at compile time some optimizations that are usually performed at run-time, by rewriting the set of dependencies before inference. This leads to avoiding the generation of irrelevant facts during the process, which is the essence of the top-down approach. We shall take these optimizations into account for the implementation.

4.3 Extension of the model

As already stated, we assumed some restrictions on the model that can be easily lifted. The reduction of EGDs to TGDs also works in this unrestricted model, which is stated in [BV84b]. Thus all the results can be extended, as they are in [CS98] for piece resolution.

There has been a renewed interest in data dependency theory with the introduction of constrained dependencies and ordered dependencies. These type of dependency can express a wide variety of constraints on the data ([BCW95]), besides generalizing most of the temporal dependencies of the taxonomy presented in [JS92]. The chase procedure has been redesigned, still in a bottom-up way, in order to deal with constrained tuple-generating dependencies ([MS96]), which are constrained functional dependencies. Our procedure can serve as a basis for the design of a top-down chase for constrained tuple-generating dependencies.

References

[AHV95] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley, Reading, Mass., 1995.

[BC90] J. Biskup and B. Convent, Relational Chase Procedures Interpreted as Resolution with Paramodulation. *Fundamenta Informaticae*, 15(8):123–138, 1991.

[BCW95] Marianne Baudinet, Jan Chomicki, and Pierre Wolper. Constraint-generating dependencies. *ICDT'95, Proceedings of the 5th International Conference on Database Theory, Prague, Czech Republic*, p 322–337, 1995.

[BDR98] J. Bailey, G. Dong, and K. Ramamohanarao. Decidability and undecidability results for the termination problem of active database rules. In *PODS '98. Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Seattle, Washington*, p 264–273, 1998.

[BFH77] Catriel Beeri, Ronald Fagin, and John H. Horward. A complete axiomatization for functional and multivalued dependencies in database relations. *SIGMOD'77, Proceedings of the International Conference on Management of Data, Toronto, Canada*, p 47–61, 1977.

[BMSU86] François Bancilhon, David Maier, Yehoshua Sagiv, and Jeffrey D. Ullman. Magic sets and other strange ways to implement logic programs. *PODS'86, Proceedings of the Fifth Symposium on Principles of Database Systems, Cambridge, Massachusetts*, p 1–16, 1986.

[BV81] Catriel Beeri and Moshe Y. Vardi. The implication problem for data dependencies. *Proceedings of the 8th Colloqium on Automata, Languages and Programming, Acre (Akko), Israel*, p 73–85, 1981.

[BV84a] C. Beeri and M. Y. Vardi. Formal systems for tuple and equality generating dependencies. *SIAM Journal on Computing*, 13(1):76–98, 1984.

[BV84b] Catriel Beeri and Moshe Y. Vardi. A proof procedure for data dependencies. *Journal of the ACM*, 31(4):718–741, 1984.

[Cod72] E. F. Codd. *Further Normalization of the Database Relational Model*. R. Rustin, Ed. Prentice-Hall, Englewood Cliffs, NJ, 1972.

[CS98] Stéphane Coulondre and Eric Salvat. Piece resolution: Towards larger perspectives. *ICCS'98, Proceedings of the Sixth International Conference on Conceptual Structures, Montpellier, France*, p 179–193, 1998.

[FV86] R. Fagin and M. Y. Vardi. The theory of data dependencies: a survey. *Mathematics of Information Processing, Proceedings of Symposia in Applied Mathematics*, 34::19–72, 1986.

[GL82] Yuri Gurevich and Harry R. Lewis. The inference problem for template dependencies. *Information and Control*, 55(1–3):69–79, 1982.

[GS98] David Genest and Eric Salvat. A platform allowing typed nested graphs: How cogito became cogitant. *ICCS'98, Proceedings of the Sixth International Conference on Conceptual Structures, Montpellier, France*, p 154–161, 1998.

[Her95] Christian Herrmann. On the undecidability of implications between embedded multivalued database dependencies. *Information and Computation*, 122(2):221–235, 1995.

[JS92] C. S. Jensen and R. T. Snodgrass. Temporal specialization. *ICDE'92, Proceedings of the International Conference on Data Engineering, Tempe, Arizona*, p 594–603, 1992.

[LL97a] Mark Levene and George Loizou. The additivity problem for functional dependencies in incomplete relations. *Acta Informatica*, 34(2):135–149, 1997.

[LL97b] Mark Levene and George Loizou. Null inclusion dependencies in relational databases. *Information and Computation*, 136(2):67–108, 1997.

[LL98] M. Levene and G. Loizou. The additivity problem for data dependencies in incomplete relational databases. *Acta Informatica*, 34(2):135–149, 1997.

[LL99] Mark Levene and George Loizou. *A Guided Tour of Relational Databases and Beyond*. Springer, London, 1999.

[Mah94] Michael Maher. Constrained dependencies. *ILPS'94, Proceedings of the Workshop on Constraints and Databases, Ithaca, New York*, 1994.

[Mah97] M. J. Maher. Constrained dependencies. *Theoretical Computer Science*, 173(1):113–149, 1997.

[MS96] M. J. Maher and D. Srivastava. Chasing constrained tuple-generating dependencies. *PODS '96, Proceedings of the Fifteenth Symposium on Principles of Database Systems, Montréal, Canada*, p 127–138, 1996.

[Ng99] Wilfred Ng. Ordered functional dependencies in relational databases. *Information Systems*, 24(7):535–554, 1999.

[NR01] R. Nieuwenhuis and A. Rubio, Paramodulation-Based Theorem Proving. *Handbook of Automated Reasoning*, Elsevier Science, Chapter 7, 371–443, 2001.

[Sci82] Edward Sciore. A complete axiomatization of full join dependencies. *Journal of the ACM*, 29(2):373–393, April 1982.

[SF00] Iztok Savnik and Peter A. Flach. Discovery of multivalued dependencies from relations. Technical Report report00135, Albert-Ludwigs-Universitaet Freiburg, Institut fuer Informatik, 2000.

[SM96] Eric Salvat and Marie-Laure Mugnier. Sound and complete forward and backward chainings of graph rules. *ICCS'96, Proceedings of the Fourth International Conference on Conceptual Structures, Sydney, Australia*, p 248–262, 1996.

[Sow84] J. F. Sowa. *Conceptual Structures: Information Processing in Mind and Machine*. Addison-Wesley, Reading, Mass., 1984.

[SU82] Fereidoon Sadri and Jeffrey D. Ullman. Template dependencies: A large class of dependencies in relational databases and its complete axiomatization. *Journal of the ACM*, 29(2):363–372, 1982.

[TS86] Hisao Tamaki and Taisuke Sato. OLD resolution with tabulation. *Proceedings of the Third International Conference on Logic Programming, London*, p 84–98, 1986.

[Ull88] Jeffrey D. Ullman. *Principles of Database and Knowledge-Base Systems. Volume I: Classical Database Systems*. Computer Science Press, 1988.

[Var84] Moshe Y. Vardi. The implication and finite implication problems for typed template dependencies. *Journal of Computer and System Sciences*, 28(1):3–28, 1984.

[Vie86] L. Vielle. Recursive axioms in deductive databases: The query-subquery approach. *Proceedings of the 1st Conference on Expert Database Systems, Charleston, South Carolina* , p 253–267, 1986.

[WC95] J. Widom and S. Ceri. *Active Datbase Systems*. Morgan-Kaufmann, San Mateo, Calif., 1995.

Appendix

We now prove theorem 1 :

Proof of theorem 1

Proof. (If). Let $K \in SAT(D)$ be a relation. Let k be a valuation such that $k(J) \subseteq K$. So, by construction, $K \in SAT(\{< u, \emptyset > | u \in k(J)\})$. It follows that $K \in SAT(D_k)$. As D does not contain constants, the only difference between D_k and D_ω lies in the constants respectively introduced by the valuation k and ω. Therefore, we can construct a relation K_ω such that $K_\omega \in SAT(D_\omega)$ and K is obtained from K_ω by renaming constants of $\omega(J)$ in order to match with $k(J)$. Let k' be an extension of k on J' such that k' is the identity on $VAL(J') - VAL(J)$. By construction, as there is a valuation h on $\omega'(J')$ such that $h(\omega'(J')) \in K_\omega$, there is a valuation h' on $k'(J')$ identical to h on $VAL(J') - VAL(J)$, such that we have $h'(k'(J')) \in K$. Hence $K \in SAT(d)$, and $D \models d$.

(Only If). Let $K \in SAT(D_\omega)$ be a relation. (i) There is a valuation k on $\omega(J)$ such that $k(\omega(J)) \subseteq K$. As $\omega(J)$ contains only constants, then $\omega(J) \subseteq K$. (ii) As $D \subseteq D_\omega$, then $D_\omega \models D$ and $K \in SAT(D)$. As $D \models d$, then $K \in SAT(d)$. As $\omega(J) \subseteq K$ (cf. i) and $K \in SAT(d)$, then there is a extension k' of k on $\omega'(J')$ such that $k'(\omega'(J')) \subseteq K$. Hence $K \in SAT(Q)$, and $D_\omega \models Q$.

We now prove theorem 2. We shall first introduce several lemmas.

Lemma 1. *Let $Q =< I, \emptyset >$ and $Q' =< H, \emptyset >$ be two goals and $T =< J', J >$ be a TGD. If $Q' = \mathcal{T}(T, Q)$ then $\{Q', T\} \models Q$.*

Proof. Let $K \in SAT(\{Q', T\})$, then $K \in SAT(Q')$ and $K \in SAT(T)$. (i) By construction, there is a valuation h such that $Q' =< h(I) \backslash P \cup h(J), \emptyset >$, so there is a valuation k such that $k(h(I) \backslash P \cup h(J)) \subseteq K$. We also have $k(h(I) \backslash P) \subseteq K$, so $K \in SAT(< h(I) \backslash P, \emptyset >)$. (ii) Trivially, we have $K \in SAT(< h(J), h(J) >)$, and also $K \in SAT(< h(J), h(I) \backslash P \cup h(J) >)$. Moreover, we have $K \in SAT(T)$, thus $K \in SAT(< J', J >)$ and we also have $K \in SAT(< h(J'), h(J) >)$ and, as $P \subseteq h(J')$, $K \in SAT(< P, h(J) >)$. Therefore $K \in SAT(< P, h(I) \backslash P \cup h(J) >)$. (iii) Suppose $K \notin SAT(Q)$. Then there is not any valuation k such that $k(I) \subseteq K$. Therefore, for every valuation k such that $k(h(I) \backslash P) \subseteq K$, we have $k(P) \nsubseteq K$ (k exists because $K \in SAT(< h(I) \backslash P, \emptyset >)$ - cf. i). As there is a valuation k such that $k(h(I) \backslash P \cup h(J)) \subseteq K$ (cf. i), then $K \notin SAT(< P, h(I) \backslash P \cup h(J) >)$, which leads to a contradiction (cf. ii). So $K \in SAT(Q)$ and $\{Q', T\} \models Q$.

Lemma 2. *Let Q and Q' be two goals and T be a TGD. If $\{Q', T\} \models Q$, but $Q' \not\models Q$, then there is a rule application $B = \mathcal{T}(T, Q)$, such that $Q' \models B$.*

Proof. For this proof, we shall swap to the first-order logic framework. Φ will denote, for a given TGD, the equivalent formula in FOL. $\{Q', T\} \models Q$, therefore the set $\{\Phi(Q'), \Phi(T), \neg\Phi(Q)\}$ is unconsistant. SLD-Resolution allows to derive the empty clause. We are then sure that there is a linear refutation, starting from

the negative clause corresponding to $\neg\Phi(Q)$, provided that $\Phi(Q')$ and $\Phi(T)$ are under clausal form with exactly on positive litteral.

To do that, we must rewrite $\Phi(Q')$, $\Phi(T)$ and $\neg\Phi(Q)$ under clausal form, in the following way:

• $\Phi(Q')$ is under the form $\Phi(Q') = \exists x_1 ... \exists x_h (A_1 \wedge ... \wedge A_j)$. We need to introduce Skolem constants that we shall denote $q_i, i \in [1..h]$, each of them being respectively replacing the variable x_i. We construct j clauses[1] under the form $Q'_i = A_i[q_1, ..., q_h], i \in [1..j]$.

• $\Phi(T)$ is under the form $\Phi(T) = \exists y_1 ... \exists y_k (C_1 \wedge ... \wedge C_l) \leftarrow H_1 \wedge ... \wedge H_n$, universally closed by the variables $x_1, ..., x_p$. We need to introduce Skolem functions that we shall denote $f_i(x_1, ..., x_p), i \in [1..k]$, each of them being respectively replacing the variable y_i. We construct l clauses under the form $T_i = (C_i \vee \neg H_1 \vee ... \vee \neg H_n)[x_1, ..., x_p,\ f_1(x_1, ..., x_p),\ ..., f_k(x_1, ..., x_p)]$ with $i \in [1..l]$.

• $\Phi(Q)$ is under the form $Q = \exists x_1 ... \exists x_s (Q_1 \wedge ... \wedge Q_t)$. The negation of $\Phi(Q)$ is under the form $\neg Q_1 \vee ... \vee \neg Q_t$, universally closed by the variables $x_1, ..., x_s$. We construct a clause under the form $NQ = (\neg Q_1 \vee ... \vee \neg Q_t)[x_1, ..., x_s]$

The linear refutation starting with NQ exists. As NQ is composed only by negative litterals, NQ can only be resolved with clauses in which there are positive litterals, i.e. the clauses $Q'_a, a \in [1...j]$ and $T_b, b \in [1..l]$.

Let us suppose that the refutation only use the clauses $Q'_a, a \in [1..j]$. Then the clauses $T_b, b \in [1..l]$ are unnecessary, and thus $\{\Phi(Q'),\ \neg\Phi(Q)\}$ is unconsistant, thus $\Phi(Q') \models \Phi(Q)$, which is in contradiction with the hypothesis. Therefore the resolution does not only use the clauses $Q'_a, a \in [1..j]$.

Remember that $T_i = (C_i \vee \neg H_1 \vee ... \vee \neg H_n)[x_1, ..., x_p, f_1(x_1, ..., x_p), ..., f_r(x_1, ..., x_p)]$ with $i \in [1..l]$. If the refutation does not only use the clauses $Q'_a, a \in [1..j]$, then it uses at least one of the atoms $C_a, a \in [1..l]$ of the clauses $T_i, i \in [1..l]$ (which are the only other positive litterals). This step will give a resolvent containing the litterals $\neg H_1 \vee ... \vee \neg H_n$. The $f_1(x_1, ..., x_p), ..., f_r(x_1, ..., x_p)$ are Skolem functions. They correspond to existentially quantified variables of $\Phi(T)$, therefore there is a substitution between a $Q_b, b \in [1..t]$ and a C_a, $a \in [1..l]$, which does not maps from the equivalent existentially quantified variables (1st condition of the $\mathcal{T}$-rule verified) nor from the variables of H_i, $i \in [1..n]$ that are not in C_a. Therefore, within the tableaux framework, there is an equivalent valuation h that satisfied the first condition of the $\mathcal{T}$-rule (that concerns symbols of the conclusion of T not in the hypothesis, i.e. existentially quantified variables), and the second condition of the $\mathcal{T}$-rule (that concerns symbols of the hypothesis of T not in the conclusion of T, i.e. universally quantified variables appearing only in the hypothesis). Indeed, as the variables do not come into account in the substitution, we force the equivalent symbols (in the tableaux valuation) to be identical. This has no consequences, because we have supposed that no symbols have the same name in T and Q.

This resolution step has instanced the variables of NQ. Therefore litterals of NQ containing, as a term, one of these variables have them also instanced.

[1] The notation $F[t_1, ...t_k]$, where F is a formula means that terms of F (variables, constants and functions) are $t_1, ..., t_k$

Thus, as there is no function symbols nor variables in $Q'_a, a \in [1..j]$, these litterals are unified with $C_a[x_1, ..., x_p,\ f_1(x_1, ..., x_p),\ ..., f_r(x_1, ..., x_p)],\ a \in [1..n]$. We are therefore sure that there is at least one piece of the conclusion of $h(Q)$ appearing in the conclusion of $h(T)$. Thus the third condition of the $\mathcal{T}$-rule is satisfied.

Let us focus on the litterals added to the resolvent in the resolution step: $\neg H_1 \vee ... \vee \neg H_n$. There are two possibilities: either there are unified with the $C_a[x_1, ..., x_p, f_1(x_1, ..., x_p), ..., f_r(x_1, ..., x_p)], a \in [1..n]$, or with the $A_a, a \in [1..j]$. Let us suppose that at the moment they come into account within the resolution, they are unified with the $C_a[x_1, ..., x_p, f_1(x_1, ..., x_p), ..., f_r(x_1, ..., x_p)], a \in [1..n]$, then the resolvent would contain again the same atoms, because negative atoms of the clauses $T_i, i \in [1..l]$ are the same in each clause $T_i, i \in [1..l]$. They are thus necessarily unified with the $A_a, a \in [1..j]$, for the resolution to have an end (which is the case). Thus we see that negative litterals of the clauses $T_i, i \in [1..l]$ can be only unified with the $A_a, a \in [1..j]$. Therefore, the negation of the new goal has a refutation which does not need the clauses $T_b, b \in [1..l]$, and thus $\{\Phi(Q'), \neg\Phi(B)\}$ is unconsistant, and $Q' \models B$

Lemma 3. *Let Q be a goal and $\Gamma = \{T_1, ..., T_n\}$ be a set of TGDs. If $\Gamma \models Q$, then there is a finite sequence of indices $i_1, ..., i_p$ such that $\{T_{i_1}, ..., T_{i_p}\} \models Q$, with $i_j \in [1..n]$ and such that $\forall j \in [1..p]$, there is a goal B_{j-1} such that $\{B_{j-1}, T_{i_j}\} \models B_j$ with $B_0 =< \emptyset, \emptyset >$ and $B_p = Q$.*

Proof. By induction on p. At step 1, $T_{i_1} \models Q$. Thus $\{< \emptyset, \emptyset >, T_{i_1}\} \models Q$. By the lemma, there is a rule application between Q and T_{i_1}, giving a goal B_1 such that $< \emptyset, \emptyset > \models B_1$. Let us suppose this induction true until step $p-1$: there is a finite sequence of indices $i_1, ..., i_{p-1}$ such that $\{T_{i_1}, ..., T_{i_{p-1}}\} \models Q$, with $i_j \in [1..n]$ and such that $\forall j \in [1..p-1]$, there is a goal B_{j-1} such that $\{B_{j-1}, T_{i_j}\} \models B_j$ with $B_0 =< \emptyset, \emptyset >$ and $B_{p-1} = Q$. At step p, we must show that $\{B_{p-1}, T_{i_p}\} \models Q$ and that $\Gamma \models B_{p-1}$ Let T_{i_p} the TGD corresponding to the first clause interfering with Q in the SLD-resolution process described in the previous lemma. Let us show that the resolutions with T_{i_p} can be performed at the beginning of the SLD-resolution, and that $\Gamma \models B_{p-1}$. To do that, let us suppose that we can not group the resolutions with the clauses coming from T_{i_p}. Then there is necessarily a resolution with a clause coming from T_{i_p} that needs a previous resolution with a clause not coming from T_{i_p}. If it is the case, then (i) either there is a litteral of NQ that can not be unified with an atom of a clause coming from T_{i_p}, but it will be possible later (if it is not possible later, then we do not need this clause), (ii) or this litteral will appear later in a resolvent, and will be resolved with a clause coming from $T_{i_j}, j \leq p-2$. If a litteral of NQ can not be unified with an atom of the clause coming from T_{i_p}, it will not be possible later, because the resolution process does create other opportunities.

We now have a new clause resulting of a sequence of resolutions with the clauses coming from T_{i_p}. Thus this clause has a linear refutation. Let us show that this clause corresponds to a goal. To do that, let us perform the inverse transformation (i.e. clausal-form to FOL and thus tableaux). This clause only contains litterals from the original goal, and litterals coming from the hypothesis

of T_{i_p}, all negative. Moreover, it does not contain function symbols. Therefore we can perform the inverse transformation under the form of a goal B_{p-1} and thus $\Gamma \models B_{p-1}$.

As $\Gamma \models B_{p-1}$, by induction, there is a finite sequence of indices $i_1, ..., i_{p-1}$ such that $\{T_{i_1}, ..., T_{i_{p-1}}\} \models Q$, with $i_j \in [1..n]$ and such that $\forall j \in [1..p-1]$, there is a goal B_{j-1} such that $\{B_{j-1}, T_{i_j}\} \models B_j$ with $B_0 =< \emptyset, \emptyset >$. As we showed that $\{B_{p-1}, T_{i_p}\} \models Q$, then there is a finite sequence of indices $i_1, ..., i_p$ such that $\{T_{i_1}, ..., T_{i_p}\} \models Q$, with $i_j \in [1..n]$ and such that $\forall j \in [1..p]$, there is a goal B_{j-1} such that $\{B_{j-1}, T_{i_j}\} \models B_j$ with $B_0 =< \emptyset, \emptyset >$ and $B_p = Q$

Lemma 4. *Let Q' and Q be two goal such that $Q' \models Q$, and let $\Gamma = \{T_1, ..., T_n\}$ be a set of TGDs. If there is a sequence of rule applications starting from Q' and terminating with success, then there is a sequence of rule applications starting from Q and terminating with success.*

Proof. Let us show that if there is a rule application between Q' and a TGD T giving a new goal B', then there is also a rule application between Q and T giving a new goal B such that $B' \models B$. We then conclude by recurrence that $Q' \models Q$, thus the set $\{\Phi(Q'), \neg\Phi(Q)\}$ is unconsistant.

The SLD-resolution allows to produce the empty clause from the set of clauses corresponding to $\{\Phi(Q'), \neg\Phi(Q)\}$. There is therefore a linear refutation starting from the negative clause corresponding to $\neg\Phi(Q)$, provided that $\Phi(Q')$ be under clausal form with exactly one positive litteral. Let us perform the same transformation as in lemma 2. The linear refutation starting from NQ exists. As NQ is composed only of positive litterals, NQ can be resolved only with clauses having positive litterals, i.e. the clauses $Q'_a, a \in [1...j]$. Each litteral of NQ is thus unifiable with one of the $A_a[q_1, ..., q_h], a \in [1..j]$. All the atoms of NQ thus appear in the $A_a[q_1, ..., q_h], a \in [1..j]$ and can only differ by a variable of NQ corresponding to a term in the $A_a[q_1, ..., q_h], a \in [1..j]$. Therefore tuples of Q are a subset of the tuples of Q', and can only differ by extra symbols in Q.

If there is a rule application between Q' and a TGD T then there is a valuation h' and (at least) one piece of the conclusion of $h'(Q')$ appearing entirely in the conclusion of $h'(T)$. As tuples of Q can only have extra symbols, and as tuples of Q all appear in Q', there is also a valuation h and (at least) one piece of the conclusion of $h(Q)$ appearing entirely in the conclusion of $h(T)$. Let us construct the now goal B by removing from the conclusion of $h(Q)$ only the pieces corresponding to those removed from the conclusion of $h'(Q')$ when B' was constructed. Then the new goal B contains some tuples of $h(Q)$ that differ, by construction, from that of B' by potentially extra symbols, and some tuples of $h(T)$ that also differ from that of B by potentially extra symbols. By performing the transformation of $\neg\Phi(B)$ and $\Phi(B')$ under clausal form, it is easy to show that each litteral of the clause coming from $\neg\Phi(B)$ is unifiable with an atom of a clause coming from $\Phi(B')$, and thus that the set $\{\Phi(Q'), \neg\Phi(Q)\}$ is unconsistant, and $B' \models B$. By recurrence on the number of rule applications starting with Q', we conclude that there is also a sequence of rule applications starting from Q and terminating with success.

Proof of theorem 2

Proof. (If). By induction on the number of $\mathcal{T}$-rule applications. Trivially, as $Q_n =< \emptyset, \emptyset >$, then $D_\omega \models Q_n$. Assume the induction is true for $Q_i, \forall i \in 2, \ldots, n$, thus $D_\omega \models Q_2$. Let us prove that $D_\omega \models Q_1$. By lemma 1, as $Q_2 = \mathcal{T}(T_1, Q_1)$, then $\{Q_2, T_1\} \models Q_1$, and also $\{Q_2, D_\omega\} \models Q_1$. As $D_\omega \models Q_2$, it follows that $D_\omega \models Q_1$.

(Only If). By lemma 3, there is a sequence of indices $i_1, ..., i_p$ such that $\{T_{i_1}, ..., T_{i_p}\} \models Q$, with $i_j \in [1..n]$ and such that $\forall j \in [1..p]$, there is a goal B_{j-1} such that $\{B_{j-1}, T_{i_j}\} \models B_j$ with $B_0 =< \emptyset, \emptyset >$ and $B_p = Q$. The proof is made by induction on p. At step 1, we have $\{< \emptyset, \emptyset >, T_{i_1}\} \models B_1$, therefore by lemma 2, there is a rule application between B_1 and T_{i_1}, giving a goal B_0 such that $< \emptyset, \emptyset > \models B_0$. As $< \emptyset, \emptyset > \models < \emptyset, \emptyset >$, the resolution terminates with success. Let us suppose this hypothesis true until step $p-1$, i.e. there is a sequence of rule applications starting from B_{p-1} and terminating with success. At step p, we have $B_p = Q$. By lemma 2, there is a rule application between Q et T_{i_p}, giving a goal B such that $B_{p-1} \models B$. According to the induction hypothesis, there is a sequence of rule applications starting from B_{p-1} and terminating with success. By lemma 4, there is also a sequence of rule applications starting from B, thus from Q, terminating with success.

Proof of theorem 5

Proof. By theorem 3, $D \models d$ if and only if $D^* \models d$. D^* is a a set of TGDs. Therefore, by theorem 1, $D^* \models d$ if and only if $D^*_\omega \models Q$. It follows that $D \models d$ if and only if $D^*_\omega \models Q$. By theorem 2, we prove the existence of the sequence of $\mathcal{T}$-rules.

By theorem 4, $D \models e$ if and only if $D^* \models e_1$ and there is a non-trivial A-EGD in D. D^* is a a set of TGDs. Therefore, by theorem 1, $D^* \models e_1$ if and only if $D^*_\omega \models Q$. It follows that $D \models e$ if and only if $D^*_\omega \models Q$ and there is a non-trivial A-EGD in D. By theorem 2, we prove the existence of the sequence of $\mathcal{T}$-rules.