



# Validity Queries and Completeness Queries

Robert Demolombe

CERT/ONERA, 2 Avenue E. Belin BP 4025, 31055 Toulouse, France

**Abstract.** The concepts of validity and completeness are defined in terms of relationship between a correct representation of the real world and the representation of the real world which is stored in a database. Here a database is understood as a Relational database, that is, a set of atomic facts. It is assumed that some users can guarantee that some parts of the database are valid, and other parts are complete. From this information, for a given standard query, are characterized a subset of the answer to the standard query which is valid, and a superset of the answer which is complete. It is guaranteed that the correct answer is bounded up and down by these two sets.

We give a formal definition of these notions in the semantics. Then we present a sound axiomatics that can be used to define a Prolog program which computes answers to validity queries and completeness queries. The axiomatics is not complete because the definition of the non standard answers involves the notion of inclusion on Relational algebra formulas. If some strong restrictions on the completeness of the axiomatics are imposed the program always terminates.

**Keywords:** Intelligent Information Systems, Uncertainty, Epistemic Logic.

## 1 Introduction

In most cases it is not possible to guarantee that the overall information stored in a database is a correct representation of the real world. However there are many application where it can be guaranteed that some parts of the database are **valid**, in the sense that the information represented by these parts is true of the world, or are **complete**, in the sense that the overall information, corresponding to these parts, which is true of the world is represented in the database.

These concepts of validity and completeness can be used to enforce database integrity as Demolombe and Jones have shown in [3]. They can also be used to inform users about the validity and completeness of some parts of an answer to a given query, as it has been shown by Motro in [5, 6]. This latter information can be considered as answers to validity queries and to completeness queries, in the sense that, for a given standard query, a validity query asks what parts of the answer to the standard query are valid, and a completeness query asks what

---

\* I am quite grateful to Laurence Cholvy whose valuable comments helped me to improve the quality of this paper. This work has been partially supported by the CEC, in the context of the Basic Research Action MEDLAR 2.

parts are complete. The purpose of this paper is to propose a practical method to compute answers to these queries.

Let us take an example where a database contains information about flights which is represented by the relation schema:  $F(\#Flight, Departure-city, Arrival-city, Company, Day)$ . Assume that the information about validity is: *all the tuples in the relation  $F$  corresponding to flights whose departure city or arrival city is Paris, represent true facts of the world*, and the information about completeness is *all the true facts of the world corresponding to flights whose company is Air France are represented by a tuple in the relation  $F$* . Now, if one asks the standard query: *what are the flights from Paris to London ?*, the answer to the corresponding validity query is: *all the tuples in the answer are valid*, and the answer to the completeness query is *the answer is complete for all the tuples where the company is Air France*.

Validity queries and completeness queries raise two problems. The first one is: *how to formally represent information about the valid parts and the complete parts of the database?*, and the second one is: *how to compute answers to validity queries and to completeness queries?*.

In [4] Demolombe and Jones have proposed solutions, based on the theory of signaling acts, for validity queries.<sup>2</sup> In this approach it is assumed that validity of data stored in the database depends on the type of data and on the reliability of users who have inserted these data. That is, some users are assumed to be safe in regard to insertions of some particular data. The formalism that is used to represent this information is a combination of an epistemic logic and of a logic of actions. The context is more general than Relational databases. Indeed, a database is supposed to be any set of propositional sentences. The method to compute answers to validity queries should be based on automated deduction techniques [2], and it is well known that their efficiency is worst than Relational algebra techniques.

In [6] Motro has proposed solutions, in the context of Relational databases, where information about validity and completeness only depends on the type of data, and it is represented by tuples in meta relations<sup>3</sup> associated with each standard relation. For instance for a given relation  $r$  the presence of a tuple  $\langle a, x \rangle$  in the associated meta relation  $v.r$  about validity means that all the tuples in the relation  $r$  of the form  $\langle a, x \rangle$  are guaranteed to be true, and a tuple  $\langle y, b \rangle$  in the meta relation  $c.p$  about the completeness of relation  $p$  means that all the tuples of the form  $\langle y, b \rangle$  which satisfy  $p$  in the world are present in the representation of  $p$  in the database. Motro has defined an algebra on the  $v$ .relations and on the  $c$ .relations to compute the tuples that characterize, for a given standard query, the answer to the validity query and the answer to the completeness query. This algebra is very close to Relational algebra. The only significant difference is for the treatment of variables that appear in tuples, and it can easily and efficiently be implemented using existing techniques. However,

---

<sup>2</sup> In the referenced paper validity queries are called safety queries.

<sup>3</sup> As a matter of homogeneity with this paper, we have slightly changed the terminology used by Motro.

there is a strong limitation on the form of queries. Indeed, queries can only involve selections, projections and joins. No union and no difference is allowed. That means, in logical terms, no disjunction and no negation.

The main contribution of this paper is to give a general formal definition of the concepts of validity and completeness, when these concepts are applied to a database content, or to an answer to a query. In particular we show how these two concepts are related to each other. This is presented in sections 2 and 3.1.

Another contribution is to present an effective method to compute answers to validity queries and to completeness queries. The basic idea is to make use of the definitions of the valid views and of the complete views of the data base to transform standard queries into sets of queries whose answers are the answers to validity queries or to completeness queries. The presented transformation always terminates, but it does not always provides the complete set of transformed queries. This computation method is presented in section 3.

## 2 Representation of Information About Validity and Completeness

In this section is presented a semantic definition, in terms of extensions, of validity queries and completeness queries.

**Definition 1.** Relation schema. A relation schema  $rs$  of arity  $n$  is a relation name  $r$  and a tuple of definition domains  $\langle D_{i_1}, \dots, D_{i_n} \rangle$ . That is  $rs = \langle r, \langle D_{i_1}, \dots, D_{i_n} \rangle \rangle$ .

**Definition 2.** Relation schema extension. Let  $rs$  be the relation schema  $\langle r, \langle D_{i_1}, \dots, D_{i_n} \rangle \rangle$ , an extension of  $rs$  is a subset  $\rho$  of  $D_{i_1} \times \dots \times D_{i_n}$ .

**Definition 3.** Database schema. A database schema  $s$  is a tuple  $\langle rs_1, \dots, rs_p \rangle$  of relation schemas. That is  $s = \langle rs_1, \dots, rs_p \rangle$ .

**Definition 4.** Database schema extension. Let  $s$  be the database schema  $\langle rs_1, \dots, rs_p \rangle$ , an extension  $se$  of  $s$  is a tuple of relation schema extensions  $se = \langle \rho_1, \dots, \rho_p \rangle$  such that for every  $i$  in  $[1, p]$   $\rho_i$  is an extension of the relation schema  $rs_i$ .

**Definition 5.** Database state. Let  $s$  be a database schema, a database state  $db$  corresponding to  $s$  is an extension of  $s$ . That is  $db = \langle \rho_1, \dots, \rho_p \rangle$ .

**Definition 6.** World state. Let  $s$  be a database schema, a world state  $w$  corresponding to  $s$  is an extension of  $s$ . That is  $w = \langle \rho'_1, \dots, \rho'_p \rangle$ .

**Definition 7.** Possible states. Let  $s$  be a database schema, a set of possible states  $ST$  is a set of pairs  $\langle db, w \rangle$ , such that  $db$  is a database state associated with  $s$ , and  $w$  is a world state associated with  $s$ .

**Definition 8.** Relational Algebra language. Let  $s$  be a database schema, the corresponding Relational Algebra language RA is defined by the following rules.

- If  $r$  is a relation name in the database schema  $s$  then  $r \in \text{RA}$ .
- If  $f \in \text{RA}$  and  $g \in \text{RA}$  then  $(f \times g) \in \text{RA}$ ,  $(f \cup g) \in \text{RA}$ ,  $(f \cap g) \in \text{RA}$  and  $(f - g) \in \text{RA}$ .<sup>4</sup>
- If  $f \in \text{RA}$  and arity of  $f$  is  $n$ , then  $s_c(f) \in \text{RA}$ , where  $c$  is a boolean expression whose atoms are of the form  $i\theta j$  or  $i\theta a$ , where  $\theta$  is a comparison operator,  $i$  and  $j$  are in  $[1, n]$ , and  $a$  is in  $D_i$ .
- If  $f \in \text{RA}$  and arity of  $f$  is  $n$  and  $p = \langle i_1, \dots, i_m \rangle$ , where  $\{i_1, \dots, i_m\}$  is a subset of  $\{1, \dots, n\}$ , then  $\pi_p(f) \in \text{RA}$ .

**Definition 9.** Relational Algebra language evaluation. Let  $s$  be a database schema,  $se$  be an extension of  $s$ , and RA be a corresponding Relational Algebra language. The interpretation of a formula  $h$  of RA in  $se$  is denoted by  $h(se)$ .<sup>5</sup>

**Definition 10.** Inclusion of a formula in another formula. Let  $f$  and  $f'$  be two formulas of a Relational Algebra language RA corresponding to a database schema  $s$ . It is said that  $f'$  is included in  $f$  iff for every extension  $se$  of  $s$  we have  $f'(se) \subseteq f(se)$ . That is  $f' \subseteq f$  holds iff  $\forall se \ f'(se) \subseteq f(se)$  holds.

**Definition 11.** Valid view. Let  $s$  be a database schema and ST an associated set of possible states. Let  $f$  be a formula of a Relational Algebra language RA corresponding to  $s$ . The formula  $f$  is a valid view, and this fact is denoted by  $V(f)$ , iff for every possible state  $\langle db, w \rangle$  in ST we have  $f(db) \subseteq f(w)$ .

That is  $V(f) \stackrel{\text{def}}{=} \forall db \forall w (\langle db, w \rangle \in \text{ST} \Rightarrow f(db) \subseteq f(w))$ .

**Definition 12.** Complete view. Let  $s$  be a database schema and ST an associated set of possible states. Let  $f$  be a formula of a Relational Algebra language RA corresponding to  $s$ . The formula  $f$  is a complete view, and this fact is denoted by  $C(f)$ , iff for every possible state  $\langle db, w \rangle$  in ST we have  $f(w) \subseteq f(db)$ .

That is  $C(f) \stackrel{\text{def}}{=} \forall db \forall w (\langle db, w \rangle \in \text{ST} \Rightarrow f(w) \subseteq f(db))$ .

**Definition 13.** Valid subset and valid superset of a formula. Let  $f$  and  $f'$  be two formulas of the same Relational Algebra language. It is said that  $f'$  characterizes a valid subset of  $f$ , and this fact is denoted by  $v_{\text{inf}}(f, f')$ , iff we have  $V(f')$  and  $f' \subseteq f$ . It is said that  $f'$  characterizes a valid superset of  $f$ , and this fact is denoted by  $v_{\text{sup}}(f, f')$ , iff we have  $V(f')$  and  $f \subseteq f'$ .

That is,  $v_{\text{inf}}(f, f') \stackrel{\text{def}}{=} V(f') \wedge f' \subseteq f$ , and  $v_{\text{sup}}(f, f') \stackrel{\text{def}}{=} V(f') \wedge f \subseteq f'$ .

**Definition 14.** Complete subset and complete superset of a formula. Let  $f$  and  $f'$  be two formulas of the same Relational Algebra language. It is said that  $f'$  characterizes a complete subset of  $f$ , and this fact is denoted by  $c_{\text{inf}}(f, f')$ , iff we

<sup>4</sup> Additional constraints about domains of  $f$  and  $g$  are omitted for simplicity. They can be found in [7].

<sup>5</sup> A detailed definition of relational algebra expression evaluation can be found in [7].

have  $C(f')$  and  $f' \subseteq f$ . It is said that  $f'$  characterizes a complete superset of  $f$ , and this fact is denoted by  $c_{\text{sup}}(f, f')$ , iff we have  $C(f')$  and  $f \subseteq f'$ .

That is,  $c_{\text{inf}}(f, f') \stackrel{\text{def}}{=} C(f') \wedge f' \subseteq f$ , and  $c_{\text{sup}}(f, f') \stackrel{\text{def}}{=} C(f') \wedge f \subseteq f'$ .

**Definition 15.** Answer to a standard query. Let  $q$  be a formula of a Relational Algebra language corresponding to a database schema  $s$ . Let  $db$  be a database state corresponding to  $s$ . The answer to the query represented by the formula  $q$  is  $q(db)$ .

**Definition 16.** Meta database. Let  $s$  be a database schema. A meta database  $mdb$  associated with  $s$  is a set of valid views and complete views. That is  $mdb$  is of the form:  $\{V(f_1), \dots, V(f_n), C(g_1), \dots, C(g_m)\}$ .

**Definition 17.** Validity query. Answer to a validity query. Let  $q$  be a standard query and let  $mdb$  be a meta database associated with a database schema  $s$ . The validity query associated with  $q$  is  $v_{\text{inf}}(q, x)$ . An answer to the validity query  $v_{\text{inf}}(q, x)$  is a formula  $q'$ , of the same language as  $q$ , such that  $mdb$  implies  $v_{\text{inf}}(q, q')$  and, for every  $q''$  such that  $mdb$  implies  $v_{\text{inf}}(q, q'')$  we have  $q'' \subseteq q'$ . That is:  $(mdb \Rightarrow v_{\text{inf}}(q, q')) \wedge \forall q''((mdb \Rightarrow v_{\text{inf}}(q, q'')) \Rightarrow q'' \subseteq q')$ .

**Definition 18.** Completeness query. Answer to a completeness query. Let  $q$  be a standard query and let  $mdb$  be a meta database associated with a database schema  $s$ . The completeness query associated with  $q$  is  $c_{\text{sup}}(q, x)$ . An answer to the completeness query  $c_{\text{sup}}(q, x)$  is a formula  $q'$ , of the same language as  $q$ , such that  $mdb$  implies  $c_{\text{sup}}(q, q')$  and, for every  $q''$  such that  $mdb$  implies  $c_{\text{sup}}(q, q'')$  we have  $q' \subseteq q''$ . That is:  $(mdb \Rightarrow c_{\text{sup}}(q, q')) \wedge \forall q''((mdb \Rightarrow c_{\text{sup}}(q, q'')) \Rightarrow q' \subseteq q'')$ .

An intuitive justification for the definition of an answer  $q'$  to a validity query  $v_{\text{inf}}(q, x)$  is that its evaluation  $q'(db)$  on the database gives the largest subset of the answer  $q(db)$  to the standard query for which we can guarantee that the tuples in this subset satisfy  $q$  in the world. That is, such that we have  $q'(db) \subseteq q(db)$  and  $q'(db) \subseteq q(w)$ .

The answer to the validity query can be interpreted as the greatest lower bound of all the formulas  $q''$ , ordered by  $\subseteq$ , such that  $mdb$  implies  $v_{\text{inf}}(q, q'')$ . Indeed, if  $\{q''_1, \dots, q''_n\}$  is the non empty set of non equivalent formulas such that  $mdb$  implies  $v_{\text{inf}}(q, q''_i)$ , then we have  $q' = q''_1 \cup \dots \cup q''_n$ . For this reason sometimes in the following this answer will be denoted by  $q_{\text{glb}}$ .

An intuitive justification for the definition of an answer  $q'$  to a completeness query  $c_{\text{sup}}(q, x)$  is that its evaluation  $q'(db)$  on the database gives the smallest superset of the answer  $q(db)$  to the standard query for which we can guarantee that the tuples which satisfy  $q$  in the world are in this superset. That is, such that we have  $q(db) \subseteq q'(db)$  and  $q(w) \subseteq q'(db)$ .

The answer to the completeness query can be interpreted as the lowest upper bound of all the formulas  $q''$ , ordered by  $\subseteq$ , such that  $mdb$  implies  $c_{\text{sup}}(q, q'')$ . Indeed, if  $\{q''_1, \dots, q''_m\}$  is the non empty set of non equivalent formulas such

that  $\text{mdb}$  implies  $c_{\text{sup}}(q, q'')$ , then we have  $q' = q''_1 \cap \dots \cap q''_m$ . For this reason sometimes in the following this answer will be denoted by  $q_{\text{lub}}$ .

Another view of  $q_{\text{lub}}$  is that we are guaranteed that **all the tuples not in  $q_{\text{lub}}$  do not satisfy  $q$  in the world**. In other terms we have:  $t \notin q_{\text{lub}}(\text{db}) \Rightarrow t \notin q(\text{w})$ .

Also, another interpretation is that the correct answer  $q(\text{w})$  to the standard query is bounded up and down by  $q_{\text{lub}}(\text{db})$  and  $q_{\text{glb}}(\text{db})$ . That is, we have:  $q_{\text{glb}}(\text{db}) \subseteq q(\text{w}) \subseteq q_{\text{lub}}(\text{db})$ . This property shows the **practical interest of  $q_{\text{glb}}$  and  $q_{\text{lub}}$** .

### 3 Computation of Answers to Validity Queries and Completeness Queries

In this section is presented first an axiomatics that allows to infer answers to validity and completeness queries. This axiomatic is intended to support the validity of an associated Prolog program which computes these answers. However the axiomatics is not complete.

The intuitive reason is that the computation of answers involves testing expressions of the form  $f' \subseteq f$ , and this is an undecidable problem because it is equivalent to testing whether  $F' \rightarrow F$  is a theorem, where  $F$  and  $F'$  are formulas of first order predicate calculus respectively equivalent to  $f$  and  $f'$ . Moreover, we want that this program always terminates in a finite time. Therefore, if we impose termination, we have to abandon completeness. Nevertheless, in section 3.1 the proposition 19 gives a set of sufficient conditions that guarantee that  $f \subseteq g$  and that covers most of the current cases.

#### 3.1 Axiomatic Definition

**Proposition 19.** *Let  $f, f', g$  and  $g'$  be formulas of the same Relational Algebra language. We have the following propositions.*

- (i1)  $f \subseteq f$
- (i2)  $f \subseteq f \cup g$
- (i3)  $g \subseteq f \cup g$
- (i4)  $f \cap g \subseteq f$
- (i5)  $f \cap g \subseteq g$
- (i6)  $s_c f \subseteq f$
- (i7)  $s_c f \subseteq s_{c \vee c'} f$
- (i8)  $s_{c'} f \subseteq s_{c \vee c'} f$
- (i9)  $s_{c \wedge c'} f \subseteq s_c f$
- (i10)  $s_{c \wedge c'} f \subseteq s_{c'} f$
- (i11)  $(f' \subseteq f) \wedge (g' \subseteq g) \rightarrow (f' \cup g' \subseteq f \cup g)$
- (i12)  $(f' \subseteq f) \wedge (g' \subseteq g) \rightarrow (f' \cap g' \subseteq f \cap g)$
- (i13)  $(f' \subseteq f) \wedge (g' \subseteq g) \rightarrow (f' \times g' \subseteq f \times g)$
- (i14)  $(f' \subseteq f) \wedge (g' \subseteq g) \rightarrow (f' - g' \subseteq f - g)$

- (i15)  $(f' \subseteq f) \rightarrow (s_c f' \subseteq s_c f)$
- (i16)  $(f' \subseteq f) \rightarrow (\pi_p f' \subseteq \pi_p f)$
- (i17)  $(f \subseteq g) \wedge (g \subseteq h) \rightarrow (f \subseteq h)$

*Proof.* The proofs of propositions (i1) to (i10) are rather trivial. We only give the proof of proposition (i14), others are very similar. From the hypothesis  $f' \subseteq f$  we have: for very database extension  $se$ , if a tuple  $t$  is in  $f'(se)$  then  $t$  is in  $f(se)$ , or in formal terms:  $t \in f'(se) \Rightarrow t \in f(se)$ . In the same way from  $g \subseteq g'$  we have  $t \in g(se) \Rightarrow t \in g'(se)$ . Then, by contraposition, we have  $\neg(t \in g'(se)) \Rightarrow \neg(t \in g(se))$ . From these two consequences we infer  $t \in f'(se) \wedge \neg(t \in g'(se)) \Rightarrow t \in f(se) \wedge \neg(t \in g(se))$ , that is,  $t \in f' - g'(se) \Rightarrow t \in f - g(se)$ , and, from the definition of formulas inclusion, we have  $f' - g' \subseteq f - g$ .  $\square$

**Proposition 20.** *Let  $f$  and  $g$  be two formulas of the same Relational Algebra language. We have the following propositions.*

- (a1)  $V(f) \wedge V(g) \rightarrow V(f \cup g)$
- (a2)  $V(f) \wedge V(g) \rightarrow V(f \cap g)$
- (a3)  $V(f) \wedge V(g) \rightarrow V(f \times g)$
- (a4)  $V(f) \wedge C(g) \rightarrow V(f - g)$
- (a5)  $V(f) \rightarrow V(s_c f)$
- (a6)  $V(f) \rightarrow V(\pi_p f)$
  
- (b1)  $C(f) \wedge C(g) \rightarrow C(f \cup g)$
- (b2)  $C(f) \wedge C(g) \rightarrow C(f \cap g)$
- (b3)  $C(f) \wedge C(g) \rightarrow C(f \times g)$
- (b4)  $C(f) \wedge V(g) \rightarrow C(f - g)$
- (b5)  $C(f) \rightarrow C(s_c f)$
- (b6)  $C(f) \rightarrow C(\pi_p f)$

*Proof.* Let us consider the proof of proposition (a1). From the hypothesis  $V(f)$ , for any database state  $db$  and world state  $w$  we have: if  $t$  is a tuple in  $f(db)$  then  $t$  is in  $f(w)$ . That is  $t \in f(db) \Rightarrow t \in f(w)$ . In a similar way from the hypothesis  $V(g)$  we have  $t \in g(db) \Rightarrow t \in g(w)$ . From this two consequences we can infer  $t \in f(db) \vee t \in g(db) \Rightarrow t \in f(w) \vee t \in g(w)$ , and this is equivalent to  $t \in f \cup g(db) \Rightarrow t \in f \cup g(w)$ , and by definition of  $V$  we have  $V(f \cup g)$ .

Proofs of propositions (a2), (a3), (a5), (a6), (b1), (b2), (b3), (b5), and (b6) are very similar.

The proof of proposition (a4) is a bit different. From  $V(f)$  we have  $t \in f(db) \Rightarrow t \in f(w)$ , and from  $C(g)$  we have  $t \in g(w) \Rightarrow t \in g(db)$ , which, by contraposition, gives  $\neg(t \in g(db)) \Rightarrow \neg(t \in g(w))$ . Then we have  $t \in f(db) \wedge \neg(t \in g(db)) \Rightarrow t \in f(w) \wedge \neg(t \in g(w))$ . Therefore we have  $t \in f - g(db) \Rightarrow t \in f - g(w)$ . That is  $V(f - g)$ . The proof of proposition (b4) is very similar. The proof of proposition (b4) is of the same style.  $\square$

**Proposition21.** *Let  $f, f', g$  and  $g'$  be formulas of the same Relational Algebra language. We have the following propositions.*

$$(a7) \quad V(f) \wedge f' \subseteq f \not\rightarrow V(f')$$

$$(a8) \quad V(f) \not\rightarrow V(f \cap g)$$

$$(a9) \quad V(f \cup g) \not\rightarrow V(f)$$

$$(b7) \quad C(f) \wedge f' \subseteq f \not\rightarrow C(f')$$

$$(b8) \quad C(f) \not\rightarrow C(f \cap g)$$

$$(b9) \quad C(f \cup g) \not\rightarrow C(f)$$

*Proof.* The proofs are very similar for each proposition, we just give the proof of proposition (a8). For this, we consider the following example where  $V(f)$  holds and  $V(f \cap g)$  does not hold:  $t \in f(\text{db})$ ,  $t \in g(\text{db})$ ,  $t \in f(\text{w})$  and  $t \notin g(\text{w})$ . The reason why in this example we do not have  $V(f \cap g)$  is that we have  $t \in f \cap g(\text{db})$  and  $t \notin f \cap g(\text{w})$ .  $\square$

**Proposition22.** *Let  $f, f', g$  and  $g'$  be formulas of the same Relational Algebra language. The following rules hold.*

$$(r1) \quad V(f') \wedge (f' \subseteq f) \rightarrow v_{\text{inf}}(f, f')$$

$$(r2) \quad V(f') \wedge (f \subseteq f') \rightarrow v_{\text{sup}}(f, f')$$

$$(r3) \quad C(f') \wedge (f' \subseteq f) \rightarrow c_{\text{inf}}(f, f')$$

$$(r4) \quad C(f') \wedge (f \subseteq f') \rightarrow c_{\text{sup}}(f, f')$$

$$(r5) \quad v_{\text{inf}}(f, f') \wedge v_{\text{inf}}(g, g') \rightarrow v_{\text{inf}}(f \cup g, f' \cup g')$$

$$(r6) \quad v_{\text{sup}}(f, f') \wedge v_{\text{sup}}(g, g') \rightarrow v_{\text{sup}}(f \cup g, f' \cup g')$$

$$(r7) \quad c_{\text{inf}}(f, f') \wedge c_{\text{inf}}(g, g') \rightarrow c_{\text{inf}}(f \cup g, f' \cup g')$$

$$(r8) \quad c_{\text{sup}}(f, f') \wedge c_{\text{sup}}(g, g') \rightarrow c_{\text{sup}}(f \cup g, f' \cup g')$$

*For the operators  $\cap$  and  $\times$  we have the same rules as for  $\cup$*

$$(r9) \quad v_{\text{inf}}(f, f') \wedge c_{\text{sup}}(g, g') \rightarrow v_{\text{inf}}(f - g, f' - g')$$

$$(r10) \quad v_{\text{sup}}(f, f') \wedge c_{\text{inf}}(g, g') \rightarrow v_{\text{sup}}(f - g, f' - g')$$

$$(r11) \quad c_{\text{inf}}(f, f') \wedge v_{\text{sup}}(g, g') \rightarrow c_{\text{inf}}(f - g, f' - g')$$

$$(r12) \quad c_{\text{sup}}(f, f') \wedge v_{\text{inf}}(g, g') \rightarrow c_{\text{sup}}(f - g, f' - g')$$

$$(r13) \quad v_{\text{inf}}(f, f') \rightarrow v_{\text{inf}}(s_c f, s_c f')$$

$$(r14) \quad v_{\text{sup}}(f, f') \rightarrow v_{\text{sup}}(s_c f, s_c f')$$

$$(r15) \quad c_{\text{inf}}(f, f') \rightarrow c_{\text{inf}}(s_c f, s_c f')$$

$$(r16) \quad c_{\text{sup}}(f, f') \rightarrow c_{\text{sup}}(s_c f, s_c f')$$

*For the operator  $\pi_p$  we have the same rules as for the operator  $s_c$*

*Proof.* The proofs of all the propositions are very similar. We just present the proof of (r9). From the hypothesis  $v_{\text{inf}}(f, f')$  we have  $V(f')$ , and from  $c_{\text{sup}}(g, g')$

we have  $C(g')$ . Then, from (a4) we have  $C(f' - g')$ . From  $v_{\text{inf}}(f, f')$  we have  $f' \subseteq f$ , and from  $c_{\text{sup}}(g, g')$  we have  $g \subseteq g'$ . Then from (i14) we have  $f' - g' \subseteq f - g$ . This consequence and the fact  $C(f' - g')$  allow to infer  $v_{\text{inf}}(f - g, f' - g')$ .

### 3.2 Computational Definition

From the rules presented in proposition 22 we can easily define a Prolog program that computes for a given set of valid views and complete views, and for a given answer to a standard query  $q$ , the answers to associated validity queries and completeness queries. These answers are respectively obtained by running the program with the queries  $v_{\text{inf}}(q, x)$ , and  $c_{\text{sup}}(q, x)$ .

In the program are defined the predicates  $v_{\text{inf}}(x, y)$ ,  $v_{\text{sup}}(x, y)$ ,  $c_{\text{inf}}(x, y)$ ,  $c_{\text{sup}}(x, y)$ ,  $V(x)$ ,  $C(x)$  and  $\subseteq(x, y)$ . The predicate  $\subseteq(x, y)$  denotes  $x \subseteq y$ , and the function symbols  $\cup(x, y)$ ,  $\cap(x, y)$ ,  $\times(x, y)$ ,  $-(x, y)$ ,  $s(x, y)$ ,  $\pi(x, y)$ ,  $\vee(x, y)$ ,  $\wedge(x, y)$  and  $\neg(x)$  respectively denote  $x \cup y$ ,  $x \cap y$ ,  $x \times y$ ,  $s_x y$ ,  $\pi_x y$ ,  $x \vee y$ ,  $x \wedge y$  and  $\neg x$ .

The Prolog clauses defining the predicates  $v_{\text{inf}}(x, y)$ ,  $v_{\text{sup}}(x, y)$ ,  $c_{\text{inf}}(x, y)$ ,  $c_{\text{sup}}(x, y)$ ,  $V(x)$ ,  $C(x)$  and  $\subseteq(x, y)$  are trivial reformulations of their corresponding rules in the propositions 19, 20 and 22. For instance, we give these clauses for the predicate  $v_{\text{inf}}(x, y)$ .

$$\begin{aligned}
v_{\text{inf}}(x, y) &: - V(y), \subseteq(y, x) \\
v_{\text{inf}}(\cup(x, y), \cup(x', y')) &: - v_{\text{inf}}(x, x'), v_{\text{inf}}(y, y') \\
v_{\text{inf}}(\cap(x, y), \cap(x', y')) &: - v_{\text{inf}}(x, x'), v_{\text{inf}}(y, y') \\
v_{\text{inf}}(\times(x, y), \times(x', y')) &: - v_{\text{inf}}(x, x'), v_{\text{inf}}(y, y') \\
v_{\text{inf}}(-(x, y), -(x', y')) &: - v_{\text{inf}}(x, x'), c_{\text{sup}}(y, y') \\
v_{\text{inf}}(s(x, y), s(x, y')) &: - v_{\text{inf}}(y, y') \\
v_{\text{inf}}(\pi(x, y), \pi(x, y')) &: - v_{\text{inf}}(y, y')
\end{aligned}$$

Assumptions about valid views and complete views are represented as data of the program. For instance if it is assumed that  $V(r \cup p)$  and  $C(s_{1=a}, r)$  hold, we have in the program:

$$\begin{aligned}
V(\cup(r, p)) &: - \\
C(s(1 = a, r)) &: -
\end{aligned}$$

If we have, for example the query  $s_{1=a \vee 1=b} r$  the answer to the corresponding validity query is obtained by running the program with the query:  $v_{\text{inf}}(s(\vee(1 = a, 1 = b), r), x)$ .

The predicate  $\subseteq(x, y)$  is defined by a set of clauses which is a direct reformulation of propositions (i1) to (i16). For instance (i14) and (i15) are reformulated into:

$$\begin{aligned}
\subseteq(-(x', y'), -(x, y)) &: - \subseteq(x', x), \subseteq(y, y') \\
\subseteq(s(x, y'), s(x, y)) &: - \subseteq(y', y)
\end{aligned}$$

Notice that for program evaluations corresponding to queries of the form  $v_{\text{inf}}(q, y)$  and  $c_{\text{sup}}(q, y)$ , in clauses like  $v_{\text{inf}}(x, y) : -V(y), \subseteq(x, y)$ , the goal  $\subseteq(x, y)$  is called for  $x$  and  $y$  being instantiated by ground terms. In that case the evaluation of  $\subseteq(x, y)$  terminates if the rule (i17) is removed, because the complexity of terms  $x'$  and  $y'$  are respectively lower than the complexity of terms  $x$  and  $y$  in the recursive calls to  $\subseteq(x, y)$ . However if the rule (i17) is involved, termination is not guaranteed. The global program terminates if the evaluation of  $\subseteq(x, y)$  terminates. Indeed, for example, a call of the form  $v_{\text{inf}}(\cup(f, g), x)$  generates calls to  $v_{\text{inf}}(f, x')$  and  $v_{\text{inf}}(g, y')$ .

We see that termination can be reached by imposing strong limitations to the completeness of the program. For example, by removing the rule (i17). The definition of a good compromise between termination and completeness has to be investigated, but this is not the main topic of this paper.

## 4 Comparison with Other Works

Coming back to Motro's work, we have mentioned in the introduction that his method does not work for Relational Algebra formulas that contain either union or difference operators. We can see how we can compute answers to validity queries and to completeness queries, with the presented method, for two queries that contain a union and a difference.

Let, for instance, the meta database be  $\text{mdb} = \{V(s_{1=a}r), V(s_{2=b}p), C(r), C(s_{1=a}v_{2=b}p)\}$ .

Let us consider first the standard query:  $q = r \cup p$ . From  $\text{mdb}$  we can infer  $V((s_{1=a}r) \cup (s_{2=b}p))$ , and from proposition 19 we can infer:  $(s_{1=a}r) \cup (s_{2=b}p) \subseteq r \cup p$ . Then, we have  $q_{\text{glb}} = (s_{1=a}r) \cup (s_{2=b}p)$ .

Let us consider now the standard query:  $q = r - s_{2=b}p$ . From  $\text{mdb}$  we can infer  $V(s_{1=a}r)$  and  $C(s_{1=a}v_{2=b}p)$ , and from proposition 19 we have:  $(s_{1=a}r) - (s_{1=a}v_{2=b}p) \subseteq r - s_{2=b}p$ . Then, we have  $q_{\text{glb}} = (s_{1=a}r) - (s_{1=a}v_{2=b}p)$ . From  $\text{mdb}$  we also have  $C(r)$  and  $V(s_{2=b}p)$ . Then, we have  $c_{\text{sup}}(r, r)$  and  $v_{\text{inf}}(s_{2=b}p, s_{2=b}p)$ , and therefore we have  $q_{\text{lub}} = r - s_{2=b}p$ .

As it has been mentioned in the introduction the notion of validity has already been investigated in a logical framework in [4]. We would like to give some guidelines for a logical reconstruction of the notions of validity and completeness as they are defined in this paper.

Databases, meta databases and queries can be represented in a first order modal language with a fixed domain. In this language we have a doxastic modality  $B$  and an epistemic modality  $K$ . The intended meaning of  $B\varphi$  is "the database believes  $\varphi$ " and the intended meaning of  $K\varphi$  is "one knows  $\varphi$ ". The axiomatics of the doxastic logic is (KD) (see [1]) plus axiom schemas: (D')  $\neg B\varphi \rightarrow B(\neg\varphi)$ , and (C)  $B(\varphi \vee \psi) \rightarrow B\varphi \vee B\psi$ , which reflect the closed world assumption.

In this logical framework the notions of valid views and complete views might be defined as follows:  $V(F) \stackrel{\text{def}}{=} \forall t K(B(F(t)) \rightarrow F(t))$ , and  $C(F) \stackrel{\text{def}}{=} \forall t K(F(t) \rightarrow B(F(t)))$ , where  $F(t)$  is a formula of classical first order logic, and  $t$  is the tuple of free variables in this formula.

## 5 Conclusion

We have presented a formal representation of information about validity and completeness of parts of a database in terms of valid views and complete views. We have also presented formal definitions of validity queries and completeness queries associated with standard queries, and we have shown that the correct answer to a standard query is bounded up and down by the answers to the completeness query and to the validity query.

For the computation of answers to validity queries and completeness queries we have presented a set of axioms that allows to infer the answers. These answers are formulas of the Relational algebra, and they can be evaluated by standard techniques. We have shown how the axiomatics can be transformed into a Prolog program that computes the answers. The axiomatics is sound but it is not complete. The lack of completeness is not a so dramatic issue as it is for answers to standard queries. Its practical consequence is that correct answers are bounded by larger intervals than it would be possible to compute with a complete axiomatics.

In comparison to the approach by Motro (M) and to the approach by Demolombe and Jones (DJ), the presented approach is more general than M, since there is no restriction on the form of queries, and less general than DJ, since data are restricted to facts. From efficiency point of view, it can be implemented by combining Prolog deduction techniques and algebraic techniques. Then, it is less efficient than M which can be implemented with an extended version of Relational algebra, and it is more efficient than DJ which requires automated deduction techniques for full propositional calculus.

## References

1. B. F. Chellas. *Modal Logic: An introduction*. Cambridge University Press, 1988.
2. L. Cholvy, R. Demolombe, and A.J. Jones. Reasoning about the safety of information: from logical formalization to operational definition. In *Proc. of 8th International Symposium on Methodologies for Intelligent Systems*, 1994.
3. R. Demolombe and A. Jones. Integrity Constraints Revisited. In A. Olive, editor, *4th International Workshop on the Deductive Approach to Information Systems and Databases*. Universitat Politecnica de Barcelona, 1993.
4. R. Demolombe and A. Jones. Deriving answers to safety queries. In R. Demolombe and T. Imielinski, editors, *Nonstandard queries and answers*. Oxford University Press, 1994.
5. A. Motro. Completeness information and its application to query processing. In *Proc. of 12th International Conference on Very Large Data Bases*, 1986.
6. A. Motro. Integrity = validity + completeness. *ACM TODS*, 14(4), 1989.
7. J. D. Ullman. *Principles of Database Systems. Vol1 and Vol2*. Computer Science Press, 1988.