Model Theoretic and Proof Theoretic View of Relational Databases with Null Values: A Comparison

Hans–Joachim Klein

Institut für Informatik und Praktische Mathematik, Universität Kiel, 24098 Kiel

1 Introduction

In the traditional model theoretic view, a relational database is considered as an interpretation of its relation types. Interpretations have to be models of integrity constraints associated with the database schema. Queries in a language based on first–order calculus are evaluated under a given interpretation by assigning truth values to the formulas. In the proof theoretic view, a database is considered as a logical theory such that queries and integrity constraints become theorems which have to be inferred from the theory.

The extension of the proof theoretic approach to represent incomplete information of the type 'value at present unknown' in databases is elegant ([Rei84]) and seems to be more attractive than approaches offered by the model theoretic approach ([Cod79], [IL84]). However, since both views are equivalent in case of total databases(([Rei84]) and since semantics of incomplete databases can be defined by reference to total databases, the question arises how evaluation methods developed for both views are related.

We consider the definition of answers to queries in case of incomplete databases given by W. Lipski ([Lip84]) and R. Reiter ([Rei84, Rei86]). The efficient but incomplete evaluation algorithm of [Rei86] is shown to produce the same results as a simple interpretation rule for calculus queries using three–valued logic.

2 Preliminaries

In the proof theoretic view, each occurrence of an unknown value (null value) is represented by a constant not involved in any unique name axiom. In the model theoretic view, V–relations introduced in [IL84] correspond to this setting.

Let α be a finite set of *attributes*, $D = \{D_1, ..., D_k\}$ a set of *domains*, and $dom \mid \alpha \to D$ a mapping associating a *domain* $D_i \in D$ with every attribute of α. Let $DOM_\alpha = \bigcup_{i=1}^{k} D_i$. For each $A \in \alpha$ let V_A denote a countably infinite set of *variables* such that $V_A \cap DOM_\alpha = \emptyset$. For $A, B \in \alpha$ let $V_A \cap V_B = \emptyset$, if $dom(A) \neq dom(B)$, and $V_A = V_B$, if $dom(A) = dom(B)$. Let $V = \bigcup_{A\in\alpha} V_A$.

A *V–tuple* t on α is a mapping $t \mid \alpha \to DOM_\alpha \cup V$ with $t(A) \in dom(A) \cup V_A$ for every $A \in \alpha$. A *V–relation* P on α is a finite set of V–tuples on α.

An *incomplete database* over some database schema $\sigma = \{(id_1, \alpha_1), ..., (id_m, \alpha_m)\}$ is a set of V–relations with the following properties: there is exactly one relation for each relation type of σ and each variable $v \in V$ occurs at most once in the relations of the database.

Under the assumption that each variable represents exactly one missing value, semantics of V–relations and incomplete databases can be defined by valuations.

A *valuation* ν for a given set of variables V is a function $\nu \mid V \to DOM_\alpha$ such that $\nu(v) \in dom(A)$ for $v \in V_A, A \in \alpha$.

To simplify notation, ν is extended to DOM_α by putting $\nu(d) = d$ for every $d \in DOM_\alpha$. ν naturally extends to V–tuples and V–relations producing total tuples and total relations.

The following definition of semantics for incomplete databases with V–relations is intuitively evident because of the assumptions concerning null values.

Let $\sigma = \{(id_1, \alpha_1), ..., (id_m, \alpha_m)\}$ be a database schema and let $P_1, ..., P_m$ be the V-relations of an incomplete database *idb* over σ. The semantics of *idb* is defined as the set of all *possible databases* $\{\nu(P_1), ..., \nu(P_m)\}$ with ν a valuation on V. We call each $\nu(P_i)$ a *possible relation* to P_i, $i = 1, ..., m$.

Example: For the V–relation P on $\{A, B, C\}$ two possible relations are shown below :

P

A	B	C
1	ω_1	4
1	2	ω_2
1	ω_3	ω_4

R_1

A	B	C
1	3	4
1	2	5
1	3	6

R_2

A	B	C
1	2	4

R_1 and R_2 can be obtained from P by applying the following valuations:
$\nu(\omega_1) = \nu(\omega_3) = 3$, $\nu(\omega_2) = 5$, $\nu(\omega_4) = 6$ and
$\nu'(\omega_1) = \nu'(\omega_3) = 2$, $\nu'(\omega_2) = \nu'(\omega_4) = 4$, respectively.

In [Rei86] a proof theoretic perspective is proposed which deviates from the semantics underlying V–relations and the definitions given in [Rei80] and [Rei84]: no distinction is made between ordinary symbols and null values. This distinction, however, is necessary since it is not derivable from the uniqueness axioms (see also comments by [Lak89]). In the following definitions we refer to the definitions in [Rei80] and [Rei84] but we also make use of some simplifying notations introduced in [Rei86].

A *relational language* $R = (A, F)$ is a first–order language where A is an alphabet of symbols and F is a set of well formed formulas constructed using the symbols of A in the usual way. A contains variables, constant and predicate symbols, and the usual symbols for building first–order formulas. Since constant symbols are not interpreted, we simply speak of constants in the following. A has to meet the following properties:
1) The number of constants and predicate symbols is finite.
2) There is a distinguished symbol E among the predicate symbols of A functioning as equality.
3) Among the symbols of A there is a distinguished non–empty subset of unary predicate symbols, so–called *simple types*; they are used for modelling the domain concept.

Remark: Unlike [Rei84] we restrict types to simple types because we only consider queries with variables having a simple type. This restriction corresponds to the attachment of variables to attributes in the domain relational calculus.

In a relational language (A, F) a *term* is a variable or a constant of A. If P is an n–ary predicate symbol of A and $t_1, ..., t_n$ are terms, then $P(t_1, ..., t_n)$ is an *atomic formula.* The following abbreviations are used:

$(x/\tau)(\Phi)$ for $(\forall x)(\tau(x) \Rightarrow \Phi)$ $\quad$ $(\exists x/\tau)(\Phi)$ for $(\exists x)(\tau(x) \wedge \Phi)$
$\Phi(\mathbf{x})$ for $\Phi(x_1, ..., x_n)$ $\quad$ $(\mathbf{x})\Phi(\mathbf{x})$ for $(x_1, ..., x_n)\Phi(x_1, ..., x_n)$
$E(\mathbf{s}, \mathbf{t})$ for $E(s_1, t_1) \wedge ... \wedge E(s_n, t_n)$ $\quad$ $\neg E(\mathbf{s}, \mathbf{t})$ for $\neg E(s_1, t_1) \vee ... \vee \neg E(s_n, t_n)$

Definition: Let (A, F) be a relational language with the constants of A given as a disjunct union of two sets C and Ω. A finite subset $\Gamma \subseteq F$ is an *extended relational theory with null values* if Γ fulfills the following conditions (we omit the equality axioms since they are part of every relational theory):

1) For every n–ary predicate symbol P of A distinct from E there is exactly one formula in Γ (the *extension axiom* of P in Γ) having the following form: $(\mathbf{x})P(\mathbf{x}) \equiv E(\mathbf{x}, \mathbf{c}^{(1)}) \vee ... \vee E(\mathbf{x}, \mathbf{c}^{(r)})$, $\mathbf{c}^{(i)}$ n–tuples of constants of A. $r = 0$ is allowed; in this case the formula is $(\mathbf{x})\neg P(\mathbf{x})$.
2) For each pair of distinct constants $c, c' \in C$ there is a *unique name axiom* $\neg E(c, c')$ or $\neg E(c', c)$ in Γ.
3) Γ may contain additional axioms of the form $\neg E(c, \omega_i), \neg E(\omega_i, \omega_j)$ for $c \in C, \omega_i, \omega_j \in \Omega, i \neq j$.
4) Nothing else is contained in Γ.
Remark: Because of the extension axioms for simple types the domain closure axiom is not necessary.

The main difference between a simple relational theory and an extended relational theory is the partition of C in ordinary constants and constants which may be regarded as placehoulders for unknown values (or Skolem constants). In an extended relational theory only constants of C have to be distinguishable from each other. If an element of Ω is not involved in any inequality axiom this corresponds to a variable in an incomplete database given as a set of V–relations: a variable cannot be distinguished from any other element of the domain.

3 Query semantics

In the model theoretic view, semantics of queries against incomplete databases of the type considered here can be based on the semantics of queries in possible, i.e. total, databases. This allows to define answers to queries independent of query formulations in a concrete query language. Queries can be considered as functions mapping databases to finite structures which may be relations or tables with type fixed by the query. Let q be an arbitrary query of type β for some database schema σ and idb an incomplete database over σ. The following definition of an answer was proposed in [Lip84]:

Definition: The *closed world assumption response* to q in idb is defined as
$A_{cwa}(q, idb) =_{def} \{t \mid t\ V\text{–tuple on } \beta \wedge (\forall \nu : \nu \text{ valuation on } V)(\nu(t) \in q(\nu(idb)))\}$.

In this kind of answer only variables of the incomplete database the query is posed against can be used. This restricts its relevance in case of nonmonotone queries. In [Kln98] other definitions of answers are given allowing to represent more information in answers.

The closed world assumption response corresponds to what has been simply called an *answer* in [Rei84]. We first consider the definition of queries in the proof theoretic approach.

Definition: Let $R = (A, F)$ be a relational language. A *query* is an expression
$< x_1/\tau_1, ..., x_n/\tau_n \mid \Phi(x_1, ..., x_n) >$
with $\tau_1, ..., \tau_n$ simple types and $\Phi(x_1, ..., x_n)$ an element of the set of well formed formulas over R.

Since the number of constants in R is finite, the evaluation of formulas corresponds to the limited interpretation of calculus queries in the model theoretic view. Queries are evaluated in given states. Each state over a database schema corresponds to a finite set of axioms in the proof theoretic view.

Definition: An n–tuple $\mathbf{c} = (c_1, ..., c_n)$ of constants in R is an *answer tuple* to a query $< x_1/\tau_1, ..., x_n/\tau_n \mid \Phi(x_1, ..., x_n) >$ in a suitable database state s if $\tau_1(c_1) \wedge ... \wedge \tau_n(c_n)$ and $\Phi(\mathbf{c})$ can be inferred from the set of axioms determined by s. The *answer* to a query in s is the set of answer tuples to the query.

Because of the type checking by $\tau_1(c_1) \wedge ... \wedge \tau_n(c_n)$, all constants in an answer are constants of the given state. This corresponds to the restriction concerning variables in a closed world assumption response if all axioms involved in extension axioms for simple types are also involved in extension axioms for ordinary predicates.

4 Sound but incomplete query evaluation

In [Rei86] a sound but incomplete algorithm is proposed for computing answers to queries efficiently. The evaluation method is incomplete because of the well–known tautology problem connected with null values. A query is transformed into an algebraic expression with *primitive queries* as operands. Primitive queries have the form $< \mathbf{x}/\boldsymbol{\tau} \mid P(\mathbf{r}) >$ or $< \mathbf{x}/\boldsymbol{\tau} \mid \neg P(\mathbf{r}) >$ with P an n–ary predicate symbol and $\mathbf{r}$ an n–tuple of constants and variables such that all variables of $\mathbf{x}$ occur in $\mathbf{r}$.

Let $< \mathbf{x}/\boldsymbol{\tau} \mid P(\mathbf{r}) >$ be a primitive query for a database given by an extended relational theory. Let μ be a function assigning constants to variables of the alphabet A. The answer $\| < \mathbf{x}/\boldsymbol{\tau} \mid P(\mathbf{r}) > \|$ is defined as

$\| < \mathbf{x}/\boldsymbol{\tau} \mid P(\mathbf{r}) > \| =_{def} \{\mu(\mathbf{x}) \mid \; \mu|\mathbf{x} \rightarrow DOM_{\boldsymbol{\tau}} \wedge \mu(\mathbf{r}) \in |P|\}.$

$DOM_{\boldsymbol{\tau}}$ denotes the set of all tuples on $\boldsymbol{\tau}$ which can be constructed due to the extension axioms given for the types in $\boldsymbol{\tau}$. $|P|$ denotes the set of all tuples for P determined by the extension axiom for P.

Let $< \mathbf{x}/\boldsymbol{\tau} \mid \neg P(\mathbf{r}) >$ be a primitive query with P not denoting the equality predicate. The following holds:

$\mathbf{c} \in \| < \mathbf{x}/\boldsymbol{\tau} \mid \neg P(\mathbf{r}) > \| \Leftrightarrow_{def} \mathbf{c} \in DOM_{\boldsymbol{\tau}} \wedge \mu(\mathbf{x}) = \mathbf{c} \Rightarrow \mu(\mathbf{r}) \notin |P|.$

Queries are translated into algebraic expressions as follows:

- Φ is transformed such that in the resulting formula Φ' only the connectives $\vee, \wedge$ and $\neg$ occur and only atomic formulas are negated in Φ'.
- Φ' is transformed into an algebraic expression by using the following equations and inclusions:

$$
\begin{array}{llll}
1) & \| < \mathbf{x}/\boldsymbol{\tau} \mid \Phi_1(\mathbf{x}) \wedge \Phi_2(\mathbf{x}) > \| & = & \| < \mathbf{x}/\boldsymbol{\tau} \mid \Phi_1(\mathbf{x}) > \| \;\cap\; \| < \mathbf{x}/\boldsymbol{\tau} \mid \Phi_2(\mathbf{x}) > \| \\
2) & \| < \mathbf{x}/\boldsymbol{\tau} \mid \Phi_1(\mathbf{x}) \vee \Phi_2(\mathbf{x}) > \| & \supseteq & \| < \mathbf{x}/\boldsymbol{\tau} \mid \Phi_1(\mathbf{x}) > \| \;\cup\; \| < \mathbf{x}/\boldsymbol{\tau} \mid \Phi_2(\mathbf{x}) > \| \\
3) & \| < \mathbf{x}/\boldsymbol{\tau} \mid (y/\Theta)\Phi(\mathbf{x}, y) > \| & = & DOM_{\boldsymbol{\tau}} \quad \text{if } |\Theta| = \emptyset \\
 & \| < \mathbf{x}/\boldsymbol{\tau} \mid (y/\Theta)\Phi(\mathbf{x}, y) > \| & = & \Delta_\Theta \| < \mathbf{x}/\boldsymbol{\tau}, y/\Theta \mid \Phi(\mathbf{x}, y) > \| \quad \text{if } |\Theta| \neq \emptyset \\
4) & \| < \mathbf{x}/\boldsymbol{\tau} \mid (\exists y/\Theta)\Phi(\mathbf{x}, y) > \| & \supseteq & \Pi \| < \mathbf{x}/\boldsymbol{\tau}, y/\Theta \mid \Phi(\mathbf{x}, y) > \|
\end{array}
$$

with Δ_Θ and Π defined as follows:

$\Delta_\Theta S =_{def} \{\mathbf{a} \mid \mathbf{a}b \in S \text{ for every } b \in |\Theta|\}$; $\Pi S =_{def} \{(a_1, ..., a_{n-1}) \mid (a_1, ..., a_{n-1}, a_n) \in S\}$

5) Let y be a variable not occuring free in $\Phi(x)$;

a) $\| < y/\Theta, \mathbf{x}/\boldsymbol{\tau} \mid \Phi(\mathbf{x}) > \| = |\Theta| \times < \mathbf{x}/\boldsymbol{\tau} \mid \Phi(\mathbf{x}) > \|$

b) $\| < \mathbf{x}/\boldsymbol{\tau}, y/\Theta, \mathbf{z}/\boldsymbol{\varphi} \mid \Phi(\mathbf{x}, \mathbf{z}) > \| = \Pi_{2,..,n+1,1,n+2,..,n+k+1}(|\Theta| \times \| < \mathbf{x}/\boldsymbol{\tau}, \mathbf{z}/\boldsymbol{\varphi} \mid \Phi(\mathbf{x}, \mathbf{z}) > \|)$

with $\mathbf{x}/\boldsymbol{\tau} = x_1/\tau_1, ..., x_n/\tau_n, n \geq 1$, and $\mathbf{z}/\boldsymbol{\varphi} = z_1/\varphi_1, ..., z_k/\varphi_k, k \geq 0$; Π is the projection operation with component numbers as index.

The two inclusions indicate where completion can be lost by evaluation:

- replacement of a disjunction by a union in 2) and
- transforming of an expression with a leading existential quantifier into an expression with a projection in 4).

In both cases the following problem may show up: for some tuple with constants of Ω there is a disjunction derivable from the given theory which is minimal. In model theoretic terms this minimality means that a disjunction $\Phi_1(\mathbf{t}) \vee \Phi_2(\mathbf{t})$ is valid in every model but neither for $\Phi_1(\mathbf{t})$

nor for $\Phi_2(\mathbf{t})$ validity is given for every model. This corresponds to the tautology problem we are faced with when evaluating calculus queries as described in the following.

Limited interpretation can be applied to calculus queries with the following differences to evaluation for total databases:
- V–tuples are instances of variables.
- A range term Rx has the value *true* for an interpretation I if $I(x) \in I(R)$; it has the value *false*, if no $t \in I(R)$ exists such that there is some valuation ν with $\nu(t) = \nu(I(x))$; its value is *unknown*, otherwise.
- Comparisons of the form $\omega_i \theta \omega_i$ have the value *true* if $\theta \in \{\leq, =, \geq\}$ and *false*, otherwise. Every other comparison with variables has the value *unknown*.
- Propositional expressions are evaluated using Kleene's 3–valued logic.

This kind of interpretation produces the same result as the evaluation method described above. For the argumentation it is convenient to consider C-relations as introduced in [IL84] and to use 'lazy interpretation' to compute C–relations for calculus queries. Based on these relations, evaluation of queries can be improved without loss of efficiency. New variables have to be introduced for representing values not related to a single value in the database. This proceeding is related to the extension of answers to *indefinite answers* in the proof theoretic view ([Rei80, YC88]). However, as mentioned in [ZP97], methods for C–relations are more efficient than the manipulation of disjunctive databases.

References

[Cod79] E.F. Codd: Extending the database relational model to capture more meaning. ACM Trans. on Database Systems 4 (4), pp. 397-434, 1979

[IL84] T. Imielinski and W. Lipski: Incomplete information in relational databases. J. of the ACM 31 (4), pp. 761-791, 1984

[Kle98] H.–J. Klein: On the use of marked nulls for the evaluation of queries against incomplete relational databases. 13 p., 1998, submitted

[Lak89] V.S. Lakshmanan: Query evaluation with null values: how complex is completeness? Proc. 9th Conf. on Found. of Softw. Techn. and Theor. Comp. Science, Bangalore, LNCS 405, pp. 204-222, 1989

[Lip84] W. Lipski: On relational algebra with marked nulls. Proc. 3rd ACM PODS, pp. 201-203, 1984

[Rei80] R. Reiter: Equality and domain closure in first–order databases. J. of the ACM 27 (2), pp. 235-249, 1980

[Rei84] R. Reiter: Towards a logical reconstruction of relational database theory. In: On Conceptual Modelling: Perspectives from Artificial Intelligence, Databases, and Programming Languages (M.L. Brodie et al., eds.), Springer–Verlag, Berlin, pp. 191-233, 1984

[Rei86] R. Reiter: A sound and sometimes complete query evaluation algorithm for relational databases with null values. J. of the ACM 33 (2), pp. 349-370, 1986

[YC88] L.Y. Yuan and D.–A. Chiang: A sound and complete query evaluation algorithm for relational databases with null values. Proc. ACM SIGMOD, Chicago, ACM SIGMOD RECORD 17 (3), pp. 74-81, 1988

[ZP97] E. Zimányi and A. Pirotte: Imperfect information in relational databases. In: Uncertainty Management in Information Systems (A. Motro and P. Smets, eds.), Kluwer Academic Publ., Boston, pp. 35-87, 1997