CONFLICT RESOLUTION OF RULES ASSIGNING VALUES TO VIRTUAL ATTRIBUTES

Yannis E. Ioannidis [1]
Computer Sciences Department
University of Wisconsin
Madison, WI 53706

Timos K. Sellis [2]
Computer Science Department
University of Maryland
College Park, MD 20742

Abstract

In the majority of research work done on logic programming and deductive databases, it is assumed that the set of rules defined by the user is *consistent*, i.e., that no contradictory facts can be inferred by the rules. In this paper, we address the problem of resolving conflicts of rules that assign values to virtual attributes. We devise a general framework for the study of the problem, and we propose an approach that subsumes all previously suggested solutions. Moreover, it suggests several additional solutions, which very often capture the semantics of the data more accurately than the known approaches. Finally, we address the issue of how to index rules so that conflicts are resolved efficiently, i.e., only one of the applicable rules is processed at query time.

1. INTRODUCTION

Recently, significant effort has been put into database research related to logic programming. The basic idea has been the use of logic in a database context, not only as a sound foundation for formalizing concepts, but as a query language as well. This has led to the study of *Deductive Database Systems*. A major assumption of most such studies is that the set of rules defined by the user is *consistent*, i.e., that no contradictory facts can be inferred by the rules. For example, the following is a consistent set of rules.

r_1: All Greek CS professors in the US work on databases.
r_2: All Polish CS professors in the US work on logic.

Since being Polish and being Greek are mutually exclusive, the above set of rules cannot lead to a contradiction. The same is true of the following rule set.

r_3: All Berkeley database PhD's have worked on INGRES.
r_4: Whoever has worked on INGRES works on databases.
r_5: R.K. is a Berkeley database PhD.
r_6: R.K. works on VLSI design.

The above set of rules cannot lead to a contradiction either, because R.K. may well be working on both database systems and VLSI design. To the contrary, the set of rules shown next is not consistent.

r_7: All Berkeley database PhD's have worked on INGRES.
r_8: Whoever has worked on INGRES works *only* on databases.
r_9: R.K. is a Berkeley database PhD.
r_{10}: R.K. works *only* on VLSI design.

In particular, the rules are inconsistent because the system is unable to decide what R.K. really works on.

In this paper, we address the problem of resolving conflicts of inconsistent rules, which is rarely addressed in a database context. The paper is organized as follows. Section 2 formally introduces the inconsistency problem and describes current solutions. In Section 3, we devise a general framework for resolving conflicts, which is used in Section 4 to express several examples of conflict resolution schemes. Section 5 looks at some specific conflict resolution schemes and discusses the use of rule indexing to enhance the performance of queries. Section 6 discusses a few related issues, like recursion and more general forms of rules. Finally, Section 7 summarizes the basic contributions of this paper and discusses future directions.

2. THE PROBLEM

In this section, we elaborate on the problem of inconsistencies in rules stored in a database system. We define the types of rules considered and we analyze the situations where inconsistencies arise. Finally, we give a brief summary of the solutions used in existing systems.

2.1. Rule Model

Consider a fixed, possibly infinite, set C. A database D is a vector $D = (C_D, Q_1, ..., Q_n)$, where $C_D \subseteq C$ is a (possibly infinite) set, and for each $1 \le i \le n$, $Q_i \subseteq C_D^{a_i}$ is a *relation* of *arity* a_i. We allow infinite relations in D so that primitive relations (e.g., $=$, $\ge$)

[1] Partially supported by NSF under Grant IRI-8703592.

[2] Partially supported by NSF under Grant IRI-8719458 and by the University of Maryland Institute for Advanced Computer Studies (UMIACS).

and functions (e.g., addition) can be included in our model. Clearly, such relations and functions are directly evaluable and they are not explicitly stored. Each element of Q_i is called a *tuple*. Without loss of generality, we assume for simplicity that the constants in the database are typeless. Extending our ideas to a typed system is straightforward.

We consider rules that are equivalent to Horn clauses, i.e., they are of the form

$$Q_1(x^{(1)}) \wedge \cdots \wedge Q_k(x^{(k)}) \rightarrow Q_0(x^{(0)}), \qquad (2.1)$$

where for each i, $x^{(i)}$ is a vector of variables, constants, and functions of such. We assume that the Horn clauses are range-restricted, i.e., every variable that appears in the consequent appears in the antecedent also, under some nonprimitive (i.e., explicitly stored) relation. The following are two examples of Horn clauses.

$$\mathbf{EMP}(name, sal, age, dept, num\_kids) \wedge \qquad (2.2)$$
$$sal > 50K \wedge age < 30 \rightarrow \mathbf{WELL\_PAID}(name)$$

$$\mathbf{EMP}(name, sal, age, dept, num\_kids) \wedge \qquad (2.3)$$
$$dept = \text{"toy"} \rightarrow num\_kids = 0$$

The relation in the consequent of (2.3) is "=". Formally, we should have written "=(*num_kids*,0)", but we use the infix notation for convenience. The same convention is used for the primitive literals in the antecedents.

There are several semantics that can be used for a set of rules $\{r_i\}$ with respect to the relations of a database D. One such semantics, which is our starting point, assumes that the contents of all nonprimitive relations are explicitly stored in the database. Primitive relations are directly evaluable, so their full extent is known to the database system as well. With this semantics, rules are treated as *integrity constraints*, i.e., the database contents have to satisfy them at all times. Hence, assigning constants to the variables of a rule should either make the antecedent false or make the consequent true.

The above is not a very useful semantics in deductive database systems. Explicitly storing the contents of all nonprimitive relations in the database is undesirable. A better semantics treats some of the rules as *derivation rules*. The contents of database relations, called *intentional (virtual) relations*, can be implicitly derived by rules having those relations in their consequents. In this case, the so called *least fixpoint semantics* [VanE76, Aho79] is used, i.e., the derived contents of each relation constitute the minimum set (minimum with respect to $\subseteq$) that satisfies all the rules. The existence of such a minimum is guaranteed by the fact that only Horn clauses are considered [Tars55, Aho79].

Treating a rule as an integrity constraint or a derivation rule is not an inherent property of the rule. Some general guidelines on what the natural role of a rule is state that rules with a user-defined (nonprimitive) relation in their consequent serve better as derivation rules, whereas those with a primitive relation in their consequent serve better as integrity constraints [Nico78]. According to those guidelines (2.2) should be used as a derivation rule and (2.3) should be used as an integrity constraint. This, however, is more restrictive than necessary, since storing the contents of all attributes in a nonprimitive relation is often undesirable. By treating a rule whose consequent is of the form "variable = expression" as a derivation rule (like (2.3)), we are able to implicitly assign values to the attribute of the relation whose position is occupied by "variable" in the antecedent of the rule. (If "variable" appears in multiple relations, the rule assigns values to the corresponding attributes of all of them.) Such attributes are called *intentional (virtual) attributes.*

The least fixpoint semantics are applicable in this case as well, although in a degenerate way. Since rules are assumed to be range-restricted, for every tuple in the relation where "variable" appears, the bindings in the antecedent of the rule determine the value of "expression". For = to be satisfied, the fixpoint semantics dictate that this is the value of the appropriate attribute of the given tuple in the relation. If a tuple in the relation where "variable" appears is associated with multiple bindings in the antecedent of the rule generating different values for "variable", then inconsistency arises and must be resolved.

2.2. Inconsistent Sets of Rules

Consider a set of derivation rules $\{r_i\}$. We say that $\{r_i\}$ is *inconsistent*, if there exists a database D such that for some fact α, one can derive both α and *not*(α) from D and $\{r_i\}$. We shall investigate the situations that produce inconsistencies. As long as no negative information is contained in the database, i.e., there is no tuple known not to exist in a relation, inconsistency cannot arise: only positive information is stored explicitly, and only positive information can be derived by Horn clauses. This may lead to the false conclusion that, by only defining Horn clauses as derivation rules, no inconsistency can arise. This is not correct though, as the following classical example by Stonebraker [Ston86b] shows two Horn clauses that are clearly inconsistent:

$\mathbf{EMP}(name, title, desk) \rightarrow desk = \text{"steel"}$,
$\mathbf{EMP}(name, \text{"chairman"}, desk) \rightarrow desk = \text{"wood"}$.

The two rules offer two contradicting types for the desk of the chairman of the company. Our previous reasoning about when inconsistencies arise is still correct though. The inconsistency arises, because of the implicit negative information in the database that *not*("steel"="wood"). Negative facts on primitive relations are always implicitly part of the database, and inconsistencies are possible among derivation rules with such relations in their consequent. To the contrary, assuming that no explicit negative information is stored in the database,

no inconsistency can arise among derivation rules with user-defined relations in their consequent.

Equivalently, the two types of rules (with primitive and nonprimitive relations in their consequents) can be distinguished as follows. Let x and y be appropriately defined tuples, $q(D)$ be the set of tuples in D of the same arity as $\langle x,y\rangle$ (the tuple formed by the concatenation of x and y) that satisfy some qualification q, $\mathbf{Q}$ be a user-defined relation, $x.att$ be the attribute att of the tuple x, and g be a function from the set of tuples with the same arity as $\langle x,y\rangle$ (domain) to the set of legal values for attribute att (range). A derivation rule with a user-defined relation in its consequent defines elements of that relation and has the general form

$$\langle x,y\rangle \in q(D) \rightarrow x \in \mathbf{Q}. \qquad (2.4)$$

Since the database contains only positive facts, multiple rules that declare members of $\mathbf{Q}$ can never present a problem. To the contrary, a derivation rule that has in its consequent equality (=) defines elements of a function, i.e., values of the function on specific members of its domain, and has the general form

$$\langle x,y\rangle \in q(D) \rightarrow x.att = g(x,y).^{\dagger} \qquad (2.5)$$

A function is constrained to return a unique value for every

member of its domain. When multiple rules give values to *x.att* in their consequents, they effectively define a function in a piecewise fashion. Inconsistency arises when a tuple x satisfies the antecedents of multiple rules, and the g functions used by the rules give different values on x. It can also arise within a single rule when a tuple x satisfies the rule's antecedent in association with multiple y's, and the g function used by the rule gives different values for different y's. For every such tuple x, the system must choose a single rule among the qualifying ones and a single y to use for the value of *x.att*. There are several criteria that can be used to guide this choice. In the next subsection, we give a brief overview of the criteria used by some existing systems. In the section after that, we develop a framework that subsumes all currently used criteria and also introduces some new ones that capture the rule semantics in cases where the known solutions fail.

2.3. Current Solutions

To the best of our knowledge, the specific problem of resolving conflicts between rules deriving values for a virtual attribute has not been directly addressed in the context of deductive database systems. A similar problem, however, arises in systems supporting production rules (which can be used to imitate functional derivation rules), when from a set of qualifying rules, precisely one has to fire. Solutions to this problem could be used for the problem we address in this paper also. We are aware of two solutions currently in use, one in which the system is assumed responsible to resolve the conflict, and another in which the user is assumed responsible. The first is best exemplified by OPS5 [Forg79] and the second by POSTGRES [Ston88]. OPS5 uses an elaborate criterion to resolve conflicts, which takes into account structural properties of the rules and other properties of the data involved. These include the complexity of the antecedents of the rules, the recency of the rules in conflict, and the recency of the tuples satisfying the rules (Section 4). In case that no criterion resolves the conflict, a rule is chosen randomly. POSTGRES, which is a database system supporting production rules, uses a very simple criterion at the expense of making the rule design more complicated. Each rule is assigned a priority. When rules are in conflict, the one with the highest priority is chosen.

The semantics of inconsistent logic programs have been investigated by Kifer and Lozinskii [Kife89] and Blair and Subrahmanian [Blai88]. Both efforts present nontraditional logics that handle inconsistent beliefs. Although they deal with the semantics of such (multi-valued) logics, they do not give any insight on how a program would choose a unique value to be assigned to a virtual attribute.

Finally, similar problems arise in resolving mutliple inheritance in generalization hierarchies. Borgida addresses the problem in the form of exception handling, and defines semantics for a language that explicitly captures contradictions in an inheritance hierarchy [Borg85, Borg88].

3. A GENERAL FRAMEWORK FOR RESOLVING CONFLICTS

For the remainder of this paper, we shall use the general form of (2.5) for functional derivation rules. Consider the following set of functional derivation rules assigning values to the same attribute of a tuple (relation):

$$r_1: \quad \langle x, y_1 \rangle \in q_1(D) \rightarrow x.att = f_1(x, y_1),$$

$$r_2: \quad \langle x, y_2 \rangle \in q_2(D) \rightarrow x.att = f_2(x, y_2),$$

$$\ldots$$

$$r_m: \quad \langle x, y_m \rangle \in q_m(D) \rightarrow x.att = f_m(x, y_m).$$

In each rule, x represents a tuple from the relation whose virtual attribute is defined by the rule, whereas each y_i represents a tuple from the combination of the remaining nonprimitive relations that appear in q_i.

Every attribute of a relation can be thought of as a function from the tuples in the relation (domain) to the legal values of the attribute (range). Given a tuple as input, the function returns the value of the attribute as output. Assume that the function corresponding to attribute *att* is f. Given a tuple x, $f(x)$ has to be chosen among the $f_i(x,y)$'s for which $\langle x, y \rangle \in q_i(D)$. Each candidate value is associated with a specific rule and a specific y tuple. In the sequel, quite often we need to treat a rule with a specific y tuple as a single entity that provides a unique value for the virtual attribute. This is accomplished by partially instantiating the rule, i.e., giving specific values (from the specific y tuple) to all variables in the rule except the ones of the relation of the virtual attribute. For example, consider the rule

EMP (*name*,*salary*,*dept*) $\wedge$ **DEPT** (*dept*,*floor*,*mgr*) $\rightarrow$ *salary*=*floor* $*$ 10*K*.

Instantiating the rule with the tuple <"*toy*",4,"*Mike*"> from **DEPT** (which corresponds to the y portion of the rule) results in the following rule:

EMP (*name*,*salary*,*dept*) $\wedge$ **DEPT** (*dept*,*floor*,*mgr*) $\wedge$ *dept*="*toy*" $\wedge$ *floor*=4 $\wedge$ *mgr*="*Mike*" $\rightarrow$ *salary*=*floor* $*$ 10*K*.

Clearly, for instantiated rules, the antecedent $\langle x, y \rangle \in q_i(D)$ is satisfied by a unique y only, which has already been incorporated into q_i. In what follows, unless otherwise specified, we assume that each rule provides a unique value for the virtual attribute it defines, which therefore implies that the rules may be instantiated if necessary. Our formulation of the conflict resolution problem, however, does not depend on whether rules are instantiated or not, and we manipulate both kinds of rules similarly.

The choice of a unique value for a virtual attribute among the values returned by all qualifying rules has to be based on properties of some elements of these rules, which are the following:

(a) the rules themselves,

(b) the antecedents of the rules, i.e., the sets of tuples that satisfy the qualifications ($q_i(D)$), and

(c) the consequents of the rules, i.e., the values returned by the functions $f_i(x,y)$.

Although for every type of elements (a), (b), and (c) there exist applications requiring the use of that type as the basis of a conflict resolution criterion, we believe that their desirability increases from (a) to (c). This claim is based on the following grounds:

> *Resolution granularity*: When using (a) or (b) to resolve conflicts, the smallest granule of resolution is the rule: rules are compared and one is chosen independent of the individual tuple x. When (c) is used, the smallest granule of

† We employ the convention that the virtual attribute is always on the left-hand side of = in the consequent of functional derivation rules.

resolution is the tuple $<x,y>$: the choice is based on the final values produced for the virtual attribute, which in turn depend on the specific tuples x and y. Hence, (c) provides a finer distinction among conflicting values.

Use of declarative semantics: The mere appearance of the problem of conflict implies that the declarative semantics of a set of rules are jeopardized. This forces the use of a resolution scheme outside of the rule set, i.e., the use of a meta-rule. Nevertheless, criteria based on elements of type (a) tend to be much more procedural than those based on elements of type (b) and (c). Especially with compile-time properties of rules, criteria on elements of type (a) in effect impose a specific order of execution of rules, which is in direct opposition to the essence of declarativeness.

Amount of responsibility left to the user: Criteria based on compile-time properties of any type of elements leave much of the responsibility of resolving conflicts to the user. This in turn allows more room for errors to enter the system. Moving from elements of type (a) to elements of type (c) decreases the number of compile-time properties that one can use to resolve conflicts, and thus decreases the amount of responsibility left to the user, making the resolution strategy more robust.

Time when decision is made: The issue here is whether the decision of which rule to use in case of conflict is made at compile-time or at run-time. Elements of type (c) usually have no compile-time properties: the values given by the rules depend on the specific tuple $<x,y>$ and the state of the database. To the contrary, elements of type (b) and especially ones of type (a) have several such properties. Conflict resolution decisions made at compile time tend to be inflexible and sometimes not absolutely in accord with the desirable semantics of the rules.

For all the above reasons, we believe that criteria based on properties of the values returned by the rules for the virtual attribute are the most desirable ones. There are several examples of applications in the rest of the paper where the desirability of such criteria is demonstrated. Strangely enough, to the best of our knowledge, this is the first study that considers resolving conflicts based on the candidate values for the virtual attribute.

3.1. Resolution Algorithm

For every virtual attribute *att* in the system, there is a function u (u for *u*nique) that selects a unique value for *att* based on the values returned by all qualifying rules. In general, this is a user-defined function, based on the semantics of *att*. It is specified in the database schema, and it is independent of the number of rules assigning values to *att*. The system can have a default function u, which is applied when nothing is specified by the user.

Given a tuple x and a query on $f(x)$, i.e., $x.att$, if conflict arises, the appropriate rule elements (a), (b), and (c) are given as input to u. The relevant properties for these elements are examined, and a value is chosen for $f(x)$. That value is the output of u and the answer to the query.

The following general formula can be used for the evaluation of $f(x)$:

$$f(x) = u(\{r_i\}, \{q_i(D)\}, \{f_i(x,y)\}). \quad (3.1)$$

One can imagine any arbitrary function u being used, possibly taking into account all its inputs, i.e., all three types of elements (a), (b), and (c). Moreover, u may not necessarily choose one of the values returned by the qualifying rules, but one that combines all of them. We expect, however, that in most cases, the semantics of the data will be such that only one type of elements is used as input to u and the output value is selected among the ones returned by the qualifying rules. Also, the element properties considered by u may be dynamic or static, i.e., they may depend on the current database state or not. In the former case, u may have to access the database or some statistics about the database kept in the catalogs. In the latter case, conflicts can be resolved during a preprocessing phase, so that no access to the data is needed at query time.

In the following subsections, we examine each type of elements (a), (b), and (c) separately, giving some insights into how u may operate in each case, together with some examples intuitively falling into the specific case. For ease of presentation, we shall always assume that precisely two rules are inconsistent. Generalizing the discussion to more than two conflicting rules is straightforward. Prior to that, one more definition is necessary. Consider an indexed family of functions $\{h_i\}$. We define the function *index* as one that, given a function from $\{h_i\}$ as input, it returns its index, i.e.,

$$index(h_j) = j.$$

3.2. Resolution Criteria

3.2.1. Resolution Criteria Based on Properties of the Rules

The first type of elements whose properties can be used to resolve conflicts are the rules themselves. In this case, formula (3.1) takes on the simplified form

$$f(x) = u(\{r_i\}). \quad (3.2)$$

This criterion is already being used by existing systems [Ston88], and in our opinion, it is the least desirable. The reason is that, in addition to specifying a function u for every virtual attribute, the rule designer has to deal with properties of each rule individually, which with few exceptions would otherwise be unnecessary. Moreover, this criterion requires a good knowledge of the results of the rules in advance, at rule design time, so that their properties can be specified accordingly.

The most straightforward (and most manual) example is assigning priorities to rules, which is the hard-wired conflict resolution scheme in POSTGRES [Ston88]. Among the qualifying rules, the one with the highest priority is chosen. Formula (3.2) takes on the form

$$f(x) = u(\{r_i\}) = f_j(x,y) \text{ where} \quad (3.3)$$

$$j = index(\max_i(\{priority(r_i) \mid <x,y> \in q_i(D)\})).$$

Another rule property that might sometimes be useful in resolving conflicts is certainty factor. Several expert systems, e.g., MYCIN [Shor76], do not trust all rules alike, but they assign certainty factors to them. A reasonable way to resolve conflicts is to use the value returned by the most trustworthy rule. For systems that do assign certainty factors to rules and want to use the one with the highest certainty factor in case of conflict, formula (3.2) becomes

$$f(x) = u(\{r_i\}) = f_j(x,y) \text{ where} \quad (3.4)$$

$$j = index(\max_i(\{certainty\_factor(r_i) \mid <x,y> \in q_i(D)\})).$$

3.2.2. Resolution Criteria Based on Properties of the Antecedents of Rules

In this case, (3.1) takes on the more simplified form

$$f(x) = u(\{q_i(D)\}). \quad (3.5)$$

Function u takes as input the sets of tuples that qualify under each rule satisfied by x, and based on some properties of the sets, chooses to apply one of these rules. Some intuitive choice functions are discussed below.

Antecedent inclusion

Suppose that for all databases D, $q_1(D) \subseteq q_2(D)$, i.e., the antecedent of r_1 is strictly less general than that of r_2. This inclusion holds at the expression level, and it does not depend on the database contents. We believe that the intuition behind the two rules is that the more general one applies only when the less general one does not. Schematically, r_2 actually applies in the region between the borders of q_1 and q_2, i.e., in the region of $q_2 - q_1$ (Figure 3.1).

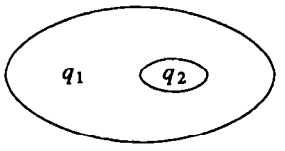

Figure 3.1: Qualification Inclusion.

One could define the two rules in a nonconflicting way by using "$q_2 \wedge not(q_1)$" as the antecedent of r_2. In the case of a chain of rules, each one of which is less general than the other, the size of the antecedent of each rule increases by a factor of two. In addition to the task of the rule designer becoming more tedious, the possibility of errors increases as well. For all these reasons, it is preferable to define the rules in a conflicting way and resolve the conflicts at a higher level.

The least general rule being the rule of choice is captured by the following u function:

$$f(x) = u(\{q_i(D)\}) = f_j(x,y) \text{ where} \quad (3.6)$$

$$j = index(\min_i(\{q_i(D) \mid \langle x,y \rangle \in q_i(D)\})).$$

In (3.6), *min* is with respect to $\subseteq$.

Alternatively, one can approach this criterion by interpreting the less general rules as exceptions to the more general ones. Normally, f_2 is used, with an exception of when q_1 holds, in which case f_1 is used. As an example, consider a rule that specifies that "all professors teach 3 semester courses per year" and another one that specifies that "the chairman teaches 1 course per year". Intuitively, the second rule is interpreted as an implicit exception to the first one, so that the chairman, who is also a professor, only teaches one course. This exception mechanism is captured by f if it is defined by (3.6).

Size minimization

Unfortunately, the above criterion cannot be applied always, since conflicts may arise between rules whose antecedents are incommensurate. A similar criterion, based on the same intuition about exceptions, uses the sizes of the qualifying sets of the two rules. The less restrictive rule, i.e., the one satisfied by more tuples in the database, applies only when the more restrictive one does not. As in the antecedent inclusion case, schematically, r_2 applies in the region of $q_2 - q_1$ (Figure 3.2).

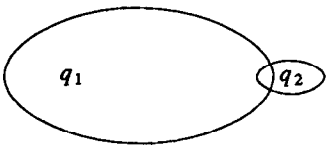

Figure 3.2: Size minimization.

Again, the main reason to define rules so that they conflict is convenience. The more restrictive rule is interpreted as an exception to the less restrictive one, and it is used whenever it is applicable. This criterion is captured by the following u function:

$$f(x) = u(\{q_i(D)\}) = f_j(x,y) \text{ where} \quad (3.7)$$

$$j = index(\min_i(\{size(q_i(D)) \mid \langle x,y \rangle \in q_i(D)\})).$$

In (3.7), *min* is with respect to integer inequality $\leq$.

As an example, taken from the Greek military conscription law, consider a rule that specifies that "all married males with one child serve in the army for 1 year" and another one that specifies that "all males who are Jehovah's witnesses serve for 4 years" (without ever carrying a gun). There are many more people with one child than there are Jehovah's witnesses in Greece. Intuitively, the second rule is interpreted as an implicit exception to the first one, so that a Jehovah's witness that has one child is required to serve for four years. This exception mechanism is captured by f if it is defined by (3.7).

Clearly, if antecedent inclusion is satisfied, then size minimization is as well. The former is a stronger requirement than the latter. In some sense, size minimization is more desirable, since it is more likely to produce a resolution to the conflict, and simultaneously less desirable, since it may fail to capture the intuition behind the rules, if the two sizes are about the same. This may be avoided if one requires that there is a substantial difference between the two sizes, by defining u appropriately.

Arbitrary

The moral one can draw from the above discussion is that there is no universal criterion that can be assumed as the default, which systems can apply without any user-provided knowledge about the semantics of the data. The qualification inclusion criterion is not always applicable, but whenever it is, it seems to be the right choice. The size minimization criterion (even in the restricted sense of significant difference in size) is neither always applicable nor always the right choice. The need for a user-defined criterion based on a general function is apparent. Any desirable semantics for the conflict resolution can be captured by such a function. Note that, for criteria on the antecedents of the rules that choose one of the f_i's as the value of f, the form of the u function is

$$f(x) = u(\{q_i(D)\}) = f_j(x,y) \text{ where}$$

$$j = index(c_i(\{h(q_i(D)) \mid \langle x,y \rangle \in q_i(D)\})),$$

where h is a function on sets of tuples, and c is a choice function based on the output of h. For qualification inclusion, c is *min* with respect to $\subseteq$ and h is the identity function. For size minimization, c is *min* with respect to integer inequality, and h is the cardinality function.

3.2.3. Resolution Criteria Based on Properties of the Consequents of Rules

Currently, conflicts are never resolved according to the values given to the virtual attribute by the rules. There are several applications, however, where the values returned by the rules are the decisive factor on which rule to apply. For example, optimization problems can be formulated as a function f (the value of the virtual attribute) being optimized according to some criterion, e.g. minimized, maximized. The rule to be used should be the one that achieves the optimal value for f. No other resolution scheme can achieve the same semantics.

For a criterion based on the values returned by the functions in the consequents of the rules, formula (3.1) takes on the simplified form

$$f(x) = u(\{f_i(x,y)\}). \tag{3.8}$$

As a concrete example, consider a car dealership that uses a rule that specifies that "all cars under 1 year old are 100% warranted" and another rule that specifies that "all cars that have been driven more than 5000 miles are 50% warranted". For a car that is less than one year old but has been driven 8000 miles, the dealer uses clauses like "whichever comes first", which can be translated to "whichever produces the least coverage". That is, conflicts are resolved based on the value of coverage percentage, and in particular by using the rule that provides the least coverage. For this example, u is min, i.e., (3.8) is instantiated into

$$coverage(x) = \min_i(\{coverage_i(x)\}).$$

Note, that a customer-chosen conflict resolution scheme would choose the rule providing the maximum coverage! This further supports our claim that only the semantics of the virtual attribute determines the correct conflict resolution scheme.

3.3. Comments

The three types of criteria specified are independent of each other. In general, it is not possible to express a choice function based on properties of some type of rule elements with a choice function based on properties of another type of elements. For example, a system that assigns priorities to rules and uses (3.3) to resolve conflicts cannot always capture the semantics of resolving conflicts by choosing the rule that gives the minimum value. In the car dealership example used in Section 3.2.3, one may argue that if one assigns higher priority to the rule that provides only 50% coverage than to the one that provides 100% coverage, conflicts are resolved by choosing the first rule as desired. If, however, the rules were of the form "all cars lose 10% of their warranty per year as they age" and "all cars lose 10% of their warranty per 5000 miles they make", it would be impossible to achieve the same conflict resolution by using priorities, simply because the rule results depend on the specific values of age and mileage of the car.

A second comment is that the previous analysis assumed that $f(x)$ is always chosen among the appropriate $f_i(x,y)$'s. This can be seen in formulas (3.3), (3.4), (3.6) and (3.7), which all have the form

$$f(x) = u(\{e_i\}) = f_j(x,y) \text{ where}$$

$$j = index(\underset{i}{c}(\{p(e_i) \mid <x,y> \in q_i(D)\})),$$

where p is the property of the elements e_i ($q_i(D)$, $f_i(x,y)$, or r_i) based on which the choice is made. Function c is the choice function, which returns one of its inputs, whose index is then returned by *index*. Thus, in some sense, in the previous analysis, one of the rules is considered to be the one that fires. Although we believe this is the most common case, the general form of u does not impose any such restrictions. For example, distributed consensus problems can be formulated with several rules assigning a value to an attribute, whose final value is determined from the proposed values according to some arbitrary criterion (e.g., taking the average of all values except the greatest and the smallest). In this case, no rule can be declared as the one that fires; in some sense, all do.

4. EXAMPLES

In this section, we present several examples where inconsistencies arise and show how they are handled by previous proposals as well as our approach.

Example 1 [Ston88]

Let **EMP** (*name*,*age*,*sal*) be a relation, where *sal* is a partial virtual attribute. Assume that the following two rules, r_1 and r_2, derive Mike's salary:

r_1: **EMP** (*name*,*age*,*sal*) $\wedge$ **EMP** (*name*1,*age*1,*sal*1)
$\wedge$ *name*="*Mike*" $\wedge$ *name*1="*Bill*" $\rightarrow$ *sal*=*sal*1
r_2: **EMP** (*name*,*age*,*sal*) $\wedge$ **EMP** (*name*1,*age*1,*sal*1)
$\wedge$ *name*="*Mike*" $\wedge$ *name*1="*Fred*" $\rightarrow$ *sal*=*sal*1

We look first at the solution POSTGRES offers for these two conflicting rules. Rules are augmented with priorities, and in the case of a conflict, the rule with the highest priority is chosen to fire. To avoid inconsistencies, r_1 is given a priority of 5, while r_2 is given a priority of 7 [Ston88]. Asking for Mike's salary returns Fred's salary.

Under our framework, both rules represent distinct points in the **EMP** $\times$ **EMP** data space, and any conflict resolution based on the size of the qualifying sets is useless. Assuming also that the salaries produced by the rules do not affect the decision of which rule is chosen, a criterion based on priorities is reasonable. Hence, the *salary* attribute is associated with the function of formula (3.3):

$$f(x) = u(\{q_i(D)\}) = f_j(x,y) \text{ where}$$

$$j = index(\underset{i}{max}(\{priority(r_i) \mid <x,y> \in q_i(D)\})).$$

The effect of such an assignment is exactly the same as with the POSTGRES rules.

Example 2 [Ston86b]

Let **EMP** (*name*,*age*,*salary*,*desk*) be a relation, where *desk* is a virtual attribute defined as follows:

r_3: **EMP** (*name*,*age*,*desk*) $\wedge$ *age*<40 $\rightarrow$ *desk*="*steel*"
r_4: **EMP** (*name*,*age*,*desk*) $\wedge$ *age*$\geq$40 $\rightarrow$ *desk*="*wood*"
r_5: **EMP** (*name*,*age*,*desk*) $\wedge$ *name*="*hotshot*" $\rightarrow$ *desk*="*wood*"
r_6: **EMP** (*name*,*age*,*desk*) $\wedge$ *name*="*bigshot*" $\rightarrow$ *desk*="*steel*"

Assuming that "hotshot" is 32 years old, there is an inconsistency in the above rules (r_5 and r_3).

For this example, Stonebraker et al. suggest to give priority 1 to r_3 and r_4, and priority 2 to r_5 and r_6 [Ston86b]. Hence, "hotshot" is assigned a wood desk, since r_5 has higher priority that r_3.

It is clear, however, that the information about "hotshot" corresponds to a fact known to the rule designer. In the EMP data space, the qualifications of the above rules are shown in Figure 4.1 (for simplicity, we only show the *age* and *name* coordinates).

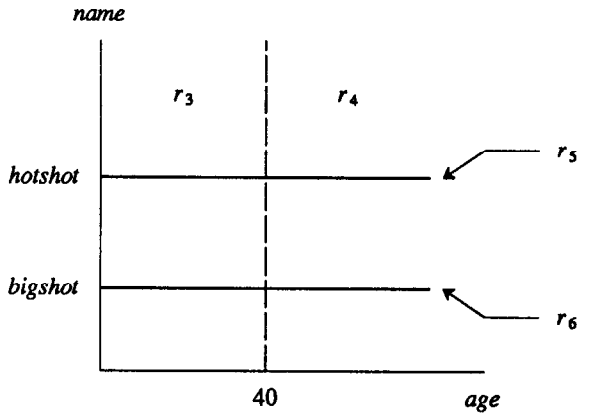

Figure 4.1: Qualifications of rules r_3 - r_6

One can see that rule r_5 corresponds to a line while rule r_3 defines a much larger area. Hence, use of priorities is unnecessary; the semantics of the rules imply that conflicts can be resolved by size minimization of the qualifying sets of tuples. The u function associated with the attribute *desk* is given by formula (3.7)

$$f(x) = u(\{q_i(D)\}) = f_j(x,y) \text{ where}$$

$$j = index(\min_i(\{size(q_i(D)) \mid \langle x,y\rangle \in q_i(D)\})),$$

where $f_i(x,y)$ and $q_i(D)$, $3 \le i \le 6$, are defined according to the consequents and antecedents of rules r_3 - r_6 respectively.

Examples 1 and 2 show that the priorities used in POSTGRES can be modeled under our framework. Moreover, there are cases where priorities are not needed (Example 2); a conflict resolution strategy based on the sizes of qualifying sets can handle such cases. Next we present a more complicated example, where a combination of various conflict resolution schemes is used.

Example 3 [Forg79]

OPS5 is a production system and uses rules of the following form:

$$x \in q(D) \rightarrow action(x).$$

Function *action* depends on the qualifying tuple x, and it can be an insertion, deletion, modification, or execution of a general procedure. Tuple insertions, deletions, and modifications make rules to fire (by satisfying their antecedent), which in turn can create a cascade of rules becoming applicable to fire. The problem of conflict resolution arises in OPS5 when multiple rules are applicable to fire. The set of rules that is applicable to fire at any one time is called the *conflict set*; a rule is in the conflict set, if there is some x in the database that makes the antecedent of the rule true. Note that the problem faced by OPS5 is not precisely one of choosing among multiple values for a virtual attribute, which is the main problem addressed in this paper. Solutions to the former problem, however, could very well serve as solutions to the latter. Thus, it is important to show that those solutions can be expressed in the general framework of Section 3 as well.

An abstraction of the conflict resolution scheme used in OPS5 uses the following criteria in the order they are given below until all but one rules are eliminated. The exact algorithm is not presented here because it makes use of details of the system that are of no interest to this paper [Forg79].

(1) choose the rule with the most recent tuples in $q(D)$
(2) choose the rule with the highest number of literals
(3) choose the rule with the highest number of constants
(4) choose the rule introduced in the conflict set most recently
(5) choose an arbitrary rule

We see that OPS5 uses a mix of time-based and size-based criteria. Rules (1) and (4) depend on the time the tuples were produced and the time the rules were put in the conflict set respectively. Rules (2) and (3) are essentially based on the sizes of the qualifying sets.

Assume that tuples are assigned timestamps when they are inserted into the database, while rules are assigned timestamps when they are put in the conflict set. The above criteria can then be modeled in our general framework as follows (for brevity we give only the criterion function, not the whole u function):

(1) : $max(time(q_i(D)))$
(2) and (3) : $min(size_i(q_i(D)))$
(4) : $max(time(r_i))$
(5) : $random(r_i)$

For a set of tuples, function *time* determines its recency from timestamps assigned to the individual tuples. For a rule, function *time* determines its recency from a timestamp assigned to the rule. Criteria (2) and (3) compare syntactic characteristics of the rules (i.e., number of literals and constants) to approximate a comparison of the sizes of the qualifying sets $q_i(D)$. Maximizing the number of literals or constants is an approximation of minimizing the size of the sets $q_i(D)$.

We can conclude that the rather complex criteria used in OPS5 for conflict resolution can be easily captured by the general framework we have suggested, thus showing the power of the mechanism. The last example that follows, is a case where the consequents of rules are used to resolve conflicts.

Example 4 [Kung86]

Some implementations of heuristic search of large graphs stored in a database have to rely on non-functional updates of virtual attributes, where user-defined semantics must be used for conflict resolution [Kung86]. Suppose we are given a map in the form of nodes and arcs between them, recorded in a relation **MAP** (*source*,*dest*,*cost*). Attribute *cost* represents the cost of going from *source* to *dest*. The problem is to find the least expensive path between two nodes "start" and "finish". This is done with the help of a relation **STATES** (*dest*,*cost*), which records the least cost of going from "start" to every node *dest* in the graph. The algorithm repeatedly updates the cost of reaching a node *dest* with the cost of reaching a neighbor node *sdest* plus the cost of going from *sdest* to *dest*. This is achieved by repeatedly using the single rule

$$\textbf{STATES}(sdest,scost) \wedge \textbf{MAP}(msource,mdest,mcost) \wedge \textbf{STATES}(dest,cost) \wedge sdest=msource \wedge mdest=dest \rightarrow cost=scost+mcost.$$

Clearly, the basic step of the algorithm introduces non-functional updates. If the node *dest* can be reached through multiple paths,

the rule tries to assign conflicting values to the cost of this node. The semantics of the update are such that the minimum cost is chosen [Kung86]. Note that in this example priorities do not work, since all cost derivation rules have the same priority. In our framework, this criterion is simply expressed by associating with the virtual attribute *cost* of **STATES** the function

$$f(x) = \min_i(\{f_i(x,y)\}),$$

where each $f_i(x,y)$ is in our case the value *mcost+scost* generated from one of the various *sdest* nodes. The above method can be actually used in any kind of search algorithm, such as A* and Branch & Bound, since it allows the user to specify in a rigorous way what the choice criterion is.

5. RULE INDEXING FOR SPECIAL CASES

In this section, we address the issue of efficiency in identifying the applicable rules producing values for a virtual attribute of a given tuple, and in resolving conflicts among multiple such rules. Given a large set of rules and a query asking for the value of a virtual attribute, identifying the relevant rules is time consuming. Several proposals have been suggested that speed up this process; some are based on putting special types of locks on the relevant attributes, tuples, and/or relations and using the locking mechanism to avoid looking at irrelevant rules [Ston88]; others are based on indexing the antecedents of rules using multidimensional data structures [Ston86a].

In this paper, we assume that some form of multidimensional index is used to index the antecedents of the rules, like an R^+-tree [Sell87]. Consider a set of rules assigning values to the same virtual attribute. The antecedents of the rules define a multidimensional space, whose dimensions are the various attributes of the relations in the antecedents. This can be further specialized into forming a space from the attributes that participate in some form of restriction in some antecedent. In this space, each antecedent specifies a region, within which the corresponding rule is applicable, like in Figures 3.1 and 3.2. Conflicts arise when two such regions from different rules overlap. Given a tuple, having an index on the antecedents allows the quick identification of the rules satisfied by the tuple. One may apply each rule, receive a value for the virtual attribute, and then choose a unique value based on a conflict resolution function u.

It is desirable that one does not try all applicable rules and then make a final choice for the attribute value, but that the index takes into account the conflict resolution scheme also, so that only one rule is processed. Unfortunately, this is not always easy or even possible, especially when u applies some nontrivial function to perform its task. There are, however, some quite common, we believe, instances of u's where indexing can help to directly identify the single rule that should be used. We specifically refer to the cases where u is of the form

$$f(x) = u(\{f_i(x,y)\}) = \underset{i}{opt}(\{f_i(x,y)\}), \qquad (5.1)$$

where *opt* is either *min* or *max*. In what follows, we shall restrict our attention to *min*. Similar things apply to *max*.

Recall that each antecedent defines a region in the multidimensional space, and that conflicts arise when these regions overlap. Taking into account (5.1) requires the following. Introduce some new dimensions in the space, corresponding to the attributes used as input to the functions in the consequents of the rules. At compile time, analyze these functions and identify the hyperplanes where the functions cross over each other. Intersecting these hyperplanes with the regions defined by the antecedents generates new regions that have no overlap among them. Every point in the space, i.e., every tuple, belongs to exactly one region, and therefore only one rule is necessary to be used to produce a value for its virtual attribute. As an example, consider the following three rules:

r_1: **EMP**(*name*,*age*,*salary*,*years*)
$\wedge\ 30 \le years \le 60 \rightarrow salary = 3K*age + 20K$

r_2: **EMP**(*name*,*age*,*salary*,*years*)
$\wedge\ 20 \le years \le 50 \rightarrow salary = 4K*age - 20K$

r_3: **EMP**(*name*,*age*,*salary*,*years*)
$\wedge\ 10 \le years \le 40 \rightarrow salary = 5K*age - 40K$

The antecedents define a 1-dimensional space (on *years*), and the regions defined by them are shown in Figure 5.1. The three functions in the consequents of the rules are plotted in Figure 5.2. The values of *age* where they cross over each other are: for f_1,f_2 is *age*=40, for f_1,f_3 is *age*=30, and for f_2,f_3 is *age*=20. Adding *age* as a dimension in the space of Figure 5.1, and taking into account the cross over points of the three functions, we have three new nonoverlapping regions, which are shown in Figure 5.3.

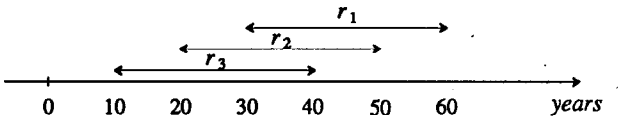

Figure 5.1: Regions of rules before conflict resolution.

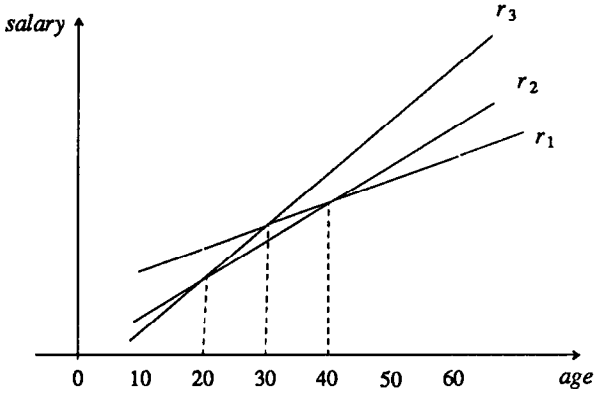

Figure 5.2: Plots of functions.

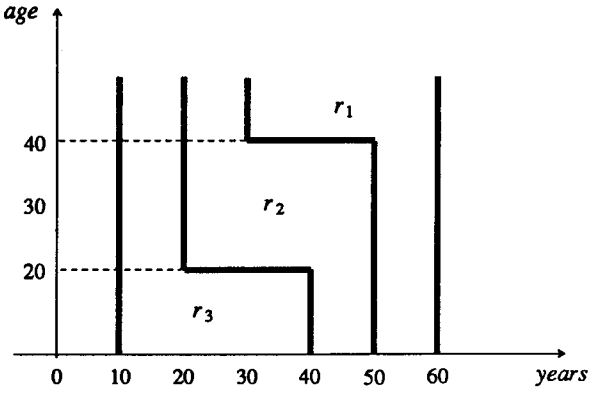

Figure 5.3: Regions of rules after conflict resolution.

Given any tuple, i.e., any specific values of *years* and *age*, there is a unique region that it falls in, i.e., there is a unique rule associated with it. This rule can be identified very fast by using an index on the nonoverlapping regions of Figure 5.3. This results in avoiding any extra work associated with trying all applicable rules and applying the conflict resolution scheme.

Note that the work of identifying the cross-over points is rather time-consuming, both because each pair of functions has to be analyzed in isolation (hence, this is quadratic in the size of the rule set) and because of the potential complexity of the functions themselves (in our example they were simple linear functions, but this is not necessarily so). The key observation, however, is that this analysis is done only once, at compile time, when the rules are defined. There is no overhead paid at query time. We expect that the amortized savings at query time make the one time cost of the analysis beneficial.

6. OTHER ISSUES

In this section, we look at several issues relevant to the basic mechanism we have suggested, like the effect of recursion and the extension to more general consequents for rules.

6.1. Recursion

Recursion in the sense that appears in nonfunctional derivation rules, i.e., the relation in the consequent appearing in the antecedent as well, is not an issue in functional derivation rules. Nevertheless, there are two different types of recursion that can arise in such rules, on which we want to elaborate. The first type, which is very unlikely to appear in any real application, arises when the function used in the rule consequent that assigns values to a virtual attribute takes the value of the virtual attribute as input. For example, the following rule gives rise to the first form of recursion:

$$\mathbf{EMP}(name, age, salary) \wedge age \leq 30$$
$$\rightarrow salary = 5*salary - 120K.$$

In general, several rules can be involved. Their consequents have the following general form (we ignore the tuples (relations) where each attribute belongs):

$$att_1 = f_1(att_1, att_2, \cdots, att_n)$$
$$att_2 = f_2(att_1, att_2, \cdots, att_n)$$
$$\cdots$$
$$att_n = f_n(att_1, att_2, \cdots, att_n)$$

The values assigned to the virtual attributes are the solutions to the above system. Assuming no interference from other rules, if the system has a unique solution, e.g., if the f_i's are multilinear in their arguments, there is no conflict to be resolved. Otherwise, the attributes' values have to be decided based on some function u. It is apparent that u must depend on properties of the rules' consequents, i.e., on the system's solution, since both the rules and the set of tuples satisfying the rules is precisely the same for each solution.

The second type of recursion arises when the virtual attribute in the consequent of a rule also appears in the antecedent of a rule in a primitive relation, i.e., constrained in some form. For example, the following rule gives rise to the second form of recursion:

$$\mathbf{EMP}(name, age, salary, num\_kids) \wedge salary \geq 40K$$
$$\rightarrow salary = 2*age - 20K.$$

The difficulty with this type of rules is that, until the rule produces a value for the virtual attribute, it is unknown if the rule is applicable. Moreover, the issue is further complicated in the case when other rules are applicable also, and a meta-rule is used to resolve conflicts. For example, assume that in addition to the previous rule, the following rule is also in effect:

$$\mathbf{EMP}(name, age, salary, num\_kids) \wedge age \leq 40$$
$$\rightarrow salary = 20K*num\_kids.$$

Assume also that conflicts are resolved by choosing the minimum value assigned to *salary*. For an EMP tuple with *age*=25 and *num_kids*=2, the second rule produces *salary*=40*K*. With this value, the first rule is applicable and produces *salary*=30*K*. The conflict is resolved by choosing 30*K* as the value of *salary*, but the rule semantics is violated, since neither of the rules is satisfied with *age*=25, *num_kids*=2, and *salary*=30*K*. It is unclear what the appropriate strategy is to avoid/break such race conditions. In the previous example, a sensible approach is to only consider the value produced by the recursive rule if its antecedent is satisfied by that value. Formalizing our intuition behind this requires further work. If we exclude such race conditions, however, such rules can be used without any problems, e.g., they can take advantage of indexing schemes like the one discussed in Section 5.

6.2. More General Forms of Rules

Finally, a natural extension of our rule model is to allow for more general consequents. In particular, we are interested in cases where ranges of values instead of single values are assigned to fields. For example, the following rules assign to a virtual field *rank* some "permissible" intervals instead of a single value

$$\mathbf{EMP}(name, age, salary, rank) \wedge age \leq 30 \rightarrow rank < 5$$
$$\mathbf{EMP}(name, age, salary, rank) \wedge salary \geq 50K \rightarrow rank > 6$$

Consistency problems arise here for employees less than 30 years old but with a salary more than 50K. The framework of Section 3 can easily cover such rules. The only complication is that conflict resolution schemes based on properties of the rule consequents manipulate sets of ranges of values. Examples of functions that can be used for that are union of ranges, intersection of ranges, maximum size range, and maximum size range on a particular dimension. The rest of the mechanism remains unchanged, including the use of indices as described in Section 5.

7. CONCLUSIONS

In this paper, the problem of modeling conflict resolution schemes for inconsistent sets of virtual attribute rules has been addressed. We have presented a general framework that captures all previously suggested solutions as well as some new ones we have proposed, thus proving its utility in a database context. We have also argued that simple solutions, such as hard-wired priorities, are not always useful, and often user-defined schemes must be specified for successful conflict resolution according to the virtual attribute semantics.

We believe that this is a first step towards a better understanding of the conflict resolution problem and the derivation of better techniques for it. As future interesting problems we view the following:

(1) The complete study of storage structures that will allow us to speed up the process of conflict resolution, based on the discussion of Section 5.

(2) The investigation of implementation techniques for user-defined conflict resolution schemes, specified on each virtual attribute at schema definition and triggered when needed.

(3) The application of these ideas to the general conflict resolution problem. For example, conflicting integrity constraints must not exist in the system. Based on our framework, one could possibly define conflicting integrity constraints, allowing only one of them to be enforced in case of conflicts.

8. REFERENCES

[Aho79]
Aho, A. and J. Ullman, "Universality of Data Retrieval Languages", in *Proc. of the 6th ACM Symposium on Principles of Programming Languages*, San Antonio, TX, January 1979, pp. 110-117.

[Blai88]
Blair, H. A., A. L. Brown, and V. S. Subrahmanian, *"A Logic Programming Semantics Scheme, Part I"*, Tech. Report, LPRG-TR-88-8, School of Computer and Information Science, Syracuse University, April 1988.

[Borg85]
Borgida, A., "Language Features for Flexible Handling of Exceptions in Information Systems", *ACM TODS* **10**, 4 (December 1985), pp. 107-131.

[Borg88]
Borgida, A., "Modeling Class Hierarchies with Contradictions", in *Proc. of the 1988 ACM-SIGMOD Conference on the Management of Data*, Chicago, IL, June 1988, pp. 434-443.

[Forg79]
Forgy, C. L., *On the Efficient Implementation of Production Systems*, Carnegie-Mellon University, Department of Computer Science, Pittsburgh, PA, PhD. dissertation, Febr. 1979.

[Kife89]
Kifer, M. and E. Lozinskii, *"RI: A Logic for Reasoning with Inconsistency"*, Proc. of Logic in Computer Science 1989 (to appear), 1989.

[Kung86]
Kung, R. et al., "Heuristic Search in Data Base Systems", in *Expert Database Systems, Proc. from the First International Workshop*, edited by L. Kerschberg, Benjamin/Cummings, Inc., Menlo Park, CA, 1986, pp. 537-548.

[Nico78]
Nicolas, J. M. and H. Gallaire, "Data Base: Theory vs. Interpretation", in *Logic and Data Bases*, edited by H. Gallaire and J. Minker, Plenum Press, New York, N.Y., 1978, pp. 33-54.

[Sell87]
Sellis, T., N. Roussopoulos, and C. Faloutsos, "The R^+-Tree: A Dynamic Index for Multi-Dimensional Objects", *Proc. 13th International VLDB Conference*, Brighton, England, September 1987, pp. 507-518.

[Shor76]
Shortliffe, E. H., *Computer-based Medical Consultations: MYCIN*, Elsevier, New York, NY, 1976.

[Ston86a]
Stonebraker, M., T. Sellis, and E. Hanson, "An Analysis of Rule Indexing Implementations in Data Base Systems", in *Proc. of the 1st International Conference on Expert Database Systems*, Charleston, SC, April 1986, pp. 353-364.

[Ston86b]
Stonebraker, M. and L. Rowe, "The Design of POSTGRES", in *Proc. of the 1986 ACM-SIGMOD Conference on the Management of Data*, Washington, DC, June 1986, pp. 340-355.

[Ston88]
Stonebraker, M., E. Hanson, and S. Potamianos, "The POSTGRES Rule Manager", *IEEE Transactions on Software Engineering* **14**, 7 (July 1988), pp. 897-907.

[Tars55]
Tarski, A., "A Lattice Theoretical Fixpoint Theorem and its Applications", *Pacific Journal of Mathematics* **5** (1955), pp. 285-309.

[VanE76]
VanEmden, M. H. and R. A. Kowalski, "The Semantics of Predicate Logic as a Programming Language", *JACM* **23**, 4 (January 1976), pp. 733-742.