Querying Database Knowledge

Amihai Motro Qiuhui Yuan

Computer Science Department
University of Southern California
Los Angeles, CA 90089-0782

Abstract

The role of database knowledge is usually limited to the evaluation of data queries In this paper we argue that when this knowledge is of substantial volume and complexity, there is genuine need to query this repository of information Moreover, since users of the database may not be able to distinguish between information that is data and information that is knowledge, access to knowledge and data should be provided with a single, coherent instrument We provide an informal review of various kinds of knowledge queries, with possible syntax and semantics We then formalize a framework of knowledge-rich databases, and a simple query language consisting of a pair of **retrieve** and **describe** statements The **retrieve** statement is for querying the data (it corresponds to the basic retrieval statement of various knowledge-rich database systems) The **describe** statement is for querying the knowledge Essentially, it inquires about the meaning of a concept under specified circumstances We provide algorithms for evaluating sound and finite knowledge answers to **describe** queries, and we demonstrate them with examples

1 Introduction

One of the most active research areas in the field of databases is the enrichment of databases with *knowledge* The representation of knowledge in databases may be regarded as the *transfer* of expertise from humans to databases

Consider a standard relational database with relation STUDENT(SNAME,GPA) that stores the grade-point averages of students, and assume that a student whose grade-point average is above 3 7 is defined to be an honor student To prepare a list of honor students, the user must know this definition of honor student, and then issue a query to list all the students whose grade-point average is at least 3 7 Assume now a knowledge-rich database with an inference rule that defines HONOR-STUDENT as a student whose grade-point average is at least 3 7 Since the definition of honor student is already present in the database, the user can issue a simpler query to list the honor students

As this example demonstrates, the principal application of knowledge in knowledge-rich databases is to allow users to formulate queries in terms that are closer to their perceptions This approach is considered superior to the traditional approach, where users applied the same knowledge *outside* the database, to translate their requests into standard database queries We note that in both approaches, the final answer consists entirely of data Knowledge was applied to *generate* the answer, but itself is not *part* of the answer

When the database knowledge is of substantial volume and complexity, there is genuine need to direct queries at this repository of information

In the database on students and their achievements, consider these two English language queries "Who are the honor students?" and "What does it take to be an honor student?" Both are natural queries, and the answers to both are included in the database However, database systems can handle the first (which applies the knowledge to access the data), but not the second (which addresses the knowledge directly)

For a second example, assume that STUDENT includes two more attributes NATIONALITY and MARITAL-STATUS, and consider these two English language queries "Are all foreign students married?" and "Must all foreign students be married?" Obviously, these are two different queries, but the user may not be aware that to a database system "Are they?" and "Must they?" are of two different *kinds* The first may be processed by

This work was supported under the AT&T Affiliates Research Program

searching for students who are foreign but not married, returning "yes" if none were found and "no" otherwise Thus, it is a simple "data query" To answer the second query the system should look for a possible item of knowledge that *requires* each foreign student to be married Again, while the information is available, the query cannot be formulated and the answer cannot be retrieved

Similarly, consider the query "Could an honor student be foreign?" To answer a query of the kind "Could it?", the system must check whether a hypothetical item of knowledge (e g , a foreign honor student) would *contradict* the stored knowledge

As a fourth example, assume that in addition to honor students, there is another category of excellence called the Dean's List, and consider this query "What is the difference between an honor student and a Dean's list student?" To answer this query the system should *compare* the definitions of these two concepts, identify their maximal shared concept, and elucidate the difference Other possible answers to such queries are that one concept is subsumed by the other, or that the concepts are unrelated

For our fifth and sixth examples, assume a database with routing information (such as airports and flights connecting them) and the standard recursive definition of reachability This database may process requests such as "List all points reachable from A", or "Give the shortest path between A and B", but not more "abstract" queries such as "Do you know how to get from any point to any other point?" or "When x is reachable from y, is it guaranteed that y is also reachable from x?" The former is a query on the availability of a definition of reachability, the latter is a query on the characteristics of the reachability relation (is it symmetric?) Neither is answerable simply by checking the current extension of the database

We note that often "knowledge queries" are follow-ups to "data queries" The answer to a query may be unexpected, raising questions about the meaning of a certain term (the first example) Or the answer to a query may seem to indicate some regularity, and the user is intrigued as to whether this regularity is indeed obligatory or only coincidental (the second example)

As these examples demonstrate, often there is genuine need to query the database knowledge Also, users who are not familiar with the internal representation of data and knowledge may not be able to distinguish easily between data queries and knowledge queries We suggest that this distinction is also undesirable Therefore, we should not only provide access to knowledge, but also unify the access to data and knowledge within a single instrument

In this paper we describe initial results from our research on the definition and evaluation of knowledge queries The work is done within the framework of first-order logic, in which relational databases are extended with definitions of inference rules Currently, the query language includes two "twin" statements, one for expressing data queries ("retrieve the data elements that satisfy ψ") and one for expressing knowledge queries ("describe the data elements that satisfy ψ") The data query corresponds to the basic retrieval statement of various knowledge-rich databases Essentially, the knowledge query statement allows users to inquire about a single concept Following are some English language equivalents of queries that are possible with this statement (the answers are based on the example knowledge base described in the paper) (1) "What is an honor student?" (answer "A student with GPA above 3 7"), (2) "When are math students whose GPA are above 3 7 eligible for teaching assistantship in the database course?" (answer "When they have completed this course under the professor currently teaching it with final grade above 3 3, or under some other professor with final grade 4 0"), and (3) "When are students who have completed a course with 4 0 eligible for teaching assistantship in this course?" (answer "When they are honor students") Additional statements are necessary to express knowledge queries that inquire about a hypothetical possibility, or compare two concepts

The problem of knowledge queries is related to the problem of *intensional answers* An intensional answer is a response to a query that uses knowledge statements that provide an abstract description of the answer Such answers provide additional meaning and insight to the standard extensional answers

Altogether, one could conceive three types of query answering mechanisms (1) the standard mechanism, that receives data queries and answers them with data, e g , [6, 11, 10, 1, 9], (2) the "intensional" mechanism, that receives data queries and answers them with knowledge, or combinations of knowledge and data, e g , [2, 8, 4, 7, 3, 12, 13], and (3) the "knowledge" mechanism, that receives knowledge queries and answers them with knowledge We are unaware of any efforts in this category (the topic of this paper)

This paper is organized as follows Section 2 defines the data model, and Section 3 defines a simple query language for data queries and knowledge queries Section 4 describes a basic algorithm for computing knowledge answers in the absence of recursion Section 5 shows how recursive rules may lead to infinite answers, and describes a modified algorithm that guarantees finiteness Section 6 concludes with a brief summary and discussion of further research issues currently under investigation

2 The Data Model

2.1 Basic Definitions

We assume a data model based on first-order logic, and our definitions are similar to those commonly used [14]

A *predicate* is a Boolean-valued function, and an *argument* is either a variable or a constant (to distinguish between variables and constants, the first character of a variable name is always a capital letter) An *atomic formula* is a predicate symbol with a list of arguments A *literal* is either an atomic formula or a negated atomic formula A *Horn clause* is a disjunction of literals with at most one positive literal Therefore, a Horn clause has one of these two forms

1 $q \leftarrow p_1 \wedge \quad \wedge p_n$
2 $\neg(p_1 \wedge \quad \wedge p_n)$

where q and each p_i are positive literals A Horn clause of the first form is a *rule* q is the *head* (or *goal*) of the rule, and $p_1 \wedge \quad \wedge p_n$ is the *body* of the rule (each p_i is a *subgoal*) Note that a rule may be without a body (i e , $n = 0$) A rule without a body and without any variables is a *fact* A Horn clause of the second form is an *integrity constraint* Here, we shall consider only Horn clauses of the first form (facts or rules) Variables appearing only in the body of a rule may be regarded as quantified existentially within the body, while other variables are quantified universally over the entire rule Thus, a rule without a body is quantified universally Finally, a conjunction of positive literals (e g , the body of a rule) will be called a *positive formula*

A knowledge-rich database D consists of

- A set P of predicates, and, for each predicate, an associated set of stored facts Each predicate is defined to be *true* over its set of associated facts, and *false* otherwise
- A set R of built-in predicates (the facts which make these predicates true are assumed to be known, and are treated as if they are stored)
- A set S of predicates, and, for each predicate, an associated set of rules Each predicate is the head of each of its associated rules

The sets of predicates P, R, and S are all mutually disjoint The first two sets will be referred to as the *extensional database* (EDB), and the third set will be referred to as the *intensional database* (IDB)

The entire database (i e , the EDB facts and the IDB rules) is understood as a collection of *axioms*, and the resolution principle is established as the *rule of inference* A fact or a rule Φ which is a logical consequence of the database (i e , follows from the axioms by the rule of inference) is a *theorem*, denoted $\vdash \Phi$ Φ is also said to be *logically derived* from the database A theorem which is fact, but not an EDB fact, is called an *intensional fact* (IDB fact) A theorem may also involve a *hypothesis* If $(p \leftarrow \varphi) \leftarrow \psi$ is a theorem [1], then $p \leftarrow \varphi$ is a theorem under the hypothesis ψ, denoted $\psi \vdash (p \leftarrow \varphi)$ $p \leftarrow \varphi$ is also said to be *logically derived* from the database and the hypothesis ψ

Given a rule $q \leftarrow p_1 \wedge \quad \wedge p_n$, the IDB predicate q is said to be *directly dependent* on the predicates $p_1, \quad , p_n$ (each p_i is either an IDB or an EDB predicate) A predicate p_0 is *dependent* on a predicate p_n, if there exist predicates $p_1, \quad , p_{n-1}$, and p_{i-1} is directly dependent on p_i, for $i = 1, \quad , n$ A rule is *recursive* if its head predicate and at least one of its body predicates are mutually dependent (i e , each depends on the other) A recursive rule is *linear*, if exactly one occurrence of its body predicates is mutually recursive with the head predicate A recursive rule is *strongly linear* if the head predicate occurs exactly once in the body of the rule [2] A rule is *typed with respect to a given predicate*, if each variable can occur only in a fixed position in any occurrence of this predicate For example, a rule that includes the occurrences $p(X, Y)$ and $p(Y, Z)$ is not typed with respect to p, and a rule that includes the occurrence $q(X, X)$ is not typed with respect to q A predicate is *recursive*, if it is the head of at least one recursive rule We shall assume that all recursive IDB predicates are defined by recursive rules that are strongly linear and typed with respect to their head predicate

2.2 Example Database

As an example, consider the following knowledge-rich database The EDB has eight predicates, as follows

$$student(Sname, Major, Gpa)$$
$$professor(Pname, Dept, Phone)$$
$$course(Ctitle, Units)$$
$$enroll(Sname, Ctitle)$$
$$teach(Pname, Ctitle)$$
$$prereq(Ctitle, Ptitle)$$
$$taught(Pname, Ctitle, Sem, Eval)$$
$$complete(Sname, Ctitle, Sem, Grade)$$

The predicates *student*, *professor*, *course*, *enroll* and *teach* have the obvious meaning (the last two pertain to the current semester) *prereq* means that course *Ptitle* is a prerequisite to course *Ctitle* *taught* means that professor *Pname* taught course *Ctitle* in semester *Sem* and received evaluation *Eval* *complete* means that student

[1] Note that this is another notation for the rule $p \leftarrow \varphi \wedge \psi$

[2] Recursive rules that are linear, but not strongly linear, can always be rewritten as strongly linear rules

Sname completed course *Ctitle* in semester *Sem* and received grade *Grade* In addition, the EDB includes the following built-in predicates $=, \neq, >, \geq, <, \leq$ The IDB has three predicates, defined as follows

$$\begin{aligned} honor(X) &\leftarrow student(X,Y,Z) \wedge (Z > 3\ 7) \\ prior(X,Y) &\leftarrow prereq(X,Y) \\ prior(X,Y) &\leftarrow prereq(X,Z) \wedge prior(Z,Y) \\ can\_ta(X,Y) &\leftarrow honor(X) \wedge complete(X,Y,Z,U) \wedge \\ &\quad (U > 3\ 3) \wedge taught(V,Y,Z,W) \wedge \\ &\quad teach(V,Y) \\ can\_ta(X,Y) &\leftarrow honor(X) \wedge complete(X,Y,Z,4\ 0) \end{aligned}$$

The predicate *honor* is defined as students with grade-point average over 3 7 The predicate *prior* is defined as the transitive closure of *prereq* The predicate *can_ta* defines the students who can be teaching assistants for a course, as honor students that had taken the course previously from the professor currently teaching this course with final grade over 3 3, as well as honor students and who had completed the course with the final grade 4 0 *prior* is the only recursive predicate, its second rule is a recursive rule, which is strongly linear and typed with respect to *prior*

3 The Query Language

The data model defined above is complemented with a query language The simple query language we describe includes two "twin" statements one for expressing data queries, and one for expressing knowledge queries

3.1 Data Queries

In most knowledge-rich database systems the language used to express knowledge statements is essentially the query language as well, and a query is a set of rules that defines the derived predicate that should be computed and printed A query may also be a predicate symbol with a list of arguments, in which case the definition of this predicate should be available to the system

Consider this query statement

retrieve p
where ψ

where p is an atomic formula (the *subject* of the query), and ψ is a positive formula (the *qualifier* of the query) ψ may not have an atomic formula of the kind $X = Y$ The variables that appear in p are assumed to be *free*, all other variables are quantified existentially

This statement finds the database values whose substitution for the variables of p and ψ satisfies $p \wedge \psi$, retrieving the values of the free variables The **where** clause may be omitted, in which case the formula ψ is assumed to be *true*

Note that when p is an EDB predicate, testing its satisfiability is done by checking a stored relation However, when p is an IDB predicate, it is defined by rules $p \leftarrow \alpha_i$, $i = 1, \quad , k$ In this case, p is satisfied, whenever one of formulas α_i is satisfied Since the predicates in ψ and in the formulas α_i may themselves be defined through rules, the processing of this statement may require recursive evaluation

It is also possible for p to be a new predicate altogether, which is then assumed to be *defined* through ψ In this case the **retrieve** statement finds the database values whose substitution for the variables of ψ satisfies ψ, retrieving the values of the free variables The user can thus avoid defining a new IDB predicate

The **retrieve** statement will be used to formulate data queries It is reminiscent of the query statement of systems such as [6, 11, 1]

Example 1: Retrieve the honor students enrolled in the databases course This data query is phrased as follows

retrieve $honor(X)$
where $enroll(X, databases)$

Example 2: Retrieve the math students whose GPA are above 3 7 and who are eligible for teaching assistantship in the databases course This data query is phrased as follows

retrieve $answer(X)$
where $can\_ta(X, databases)$ **and**
$student(X, math, V)$ **and** $(V > 3\ 7)$

Note that *answer* is not a known predicate

3.2 Knowledge Queries

As pointed out earlier, a data query applies the knowledge (i e , the rules) to generate the answer, but knowledge itself is not part of the answer Consider now this query statement

describe p
where ψ

where the subject p is an atomic formula with an IDB predicate and the qualifier ψ is as before

This statement finds theorems $p \leftarrow \varphi$, where φ is a positive formula, that are logically derived under the hypothesis ψ Note that at this point we are only concerned with the *soundness* of the answer Thus, an answer is *any* set of such theorems $p \leftarrow \varphi$ In the following sections we shall place additional restrictions on knowledge

answers and describe methods for computing these more restricted answers

The **where** clause may be omitted, in which case, since there is no hypothesis, this statement retrieves the theorems $p \leftarrow \varphi$ that are logically derived, in particular, every rule whose head predicate is p (after proper identification of the head of the rule with p)

The **describe** statement will be used to formulate knowledge queries

Example 3: When is a math student whose GPA is above 3 7 eligible for teaching assistantship in the databases course? This knowledge query is phrased as follows

describe $can\_ta(X, databases)$
where $student(X, math, V)$ **and** $(V > 3\ 7)$

The following two theorems can be logically derived under the given hypothesis, and would be included in the answer

$$can\_ta(X, databases) \leftarrow complete(X, databases, Z, U) \wedge (U > 3\ 3) \wedge taught(V, Y, Z, W) \wedge teach(V, Y)$$
$$can\_ta(X, databases) \leftarrow complete(X, databases, Z, 4\ 0)$$

The student must have completed the course under the professor currently teaching it with final grade over 3 3, or the student must have completed the course with grade 4 0

Example 4 What does it take to be an honor student? This knowledge query requires no **where** clause

describe $honor(X)$

Clearly, the definition of *honor* is a theorem that can be logically derived, and would be included in the answer

$$honor(X) \leftarrow student(X, Y, Z) \wedge (X > 3\ 7)$$

The student must have GPA over 3 7

Note that the **retrieve** statement for querying data and the **describe** statement for querying knowledge have similar syntax, differing only in their initial keyword Thus, pairs of questions such as "Retrieve the honor students" and "Describe the honor students" are expressed identically, except for the initial keyword

Finally, an answer to a knowledge query is *free of redundancies* if none of its formulas is a logical consequence of any of its other formulas An answer to a knowledge query is *complete* if no other sound and nonredundant formula exists

4 Computing Knowledge Answers in the Non-Recursive Case

In this section we describe a basic algorithm that computes sound answers to knowledge queries, in the case that the subject predicate is not recursive, and does not depend on a recursive predicate

Essentially, the algorithm constructs "derivation trees" for the subject formula In each such tree, it identifies leaf formulas that correspond to formulas of the hypothesis It then generates one answer from each tree the head is the subject predicate and the body is a conjunction of the leaf formulas that have not been identified All subtrees without hypothesis leaves are cut off below their subtree roots This ensures that answers use the most general concepts possible One exception is when the entire tree includes no hypothesis leaves In this case it constructs a derivation tree of one level This contributes a rule that corresponds to an IDB rule

Algorithm 1 Compute knowledge answers in the non-recursive case

Input IDB with rules $r_1, \quad , r_n$, a query with subject p, which is not recursive and does not depend on a recursive predicate, and qualifier (hypothesis) $\psi = h_1 \wedge \quad \wedge h_m$

Output· A set of rules $p \leftarrow \varphi$ that are logically derived under the hypothesis ψ

Figure 1 shows the algorithm in flowchart form This algorithm constructs trees of (atomic) formulas q designates the current (atomic) formula of the tree, i designates the current (atomic) formula of the hypothesis $(i = 1, \quad , m)$, and r designates the current rule under consideration $(r = 1, \quad , n)$ A single flag is used to indicate whether the rule applied at the root has so far been productive For each formula q we shall assume two variables called $hyp(q)$ and $rule(q)$ in which the current values of i and r can be saved $child(q)$ and $parent(q)$ refer to a child formula of q (the child of a leaf is nil) and to the parent formula of q (the parent of p is nil), respectively $prev(q)$ and $next(q)$ refer to the previous and next formulas, in the depth-first traversal of the tree Finally, $head(r)$ and $body(r)$ refer to the head and body, respectively, of a rule r

Initially (box 1), q is set to p, i and r are set to 0, and the flag is unset First (boxes 2–4), we attempt to identify q with one of the formulas of the hypothesis For a given formula h_i, we attempt to find a substitution function that will identify it with q If this substitution is possible (5), then it is applied to every formula of the tree If q cannot be identified with any of the formulas of the hypothesis (8), we attempt to find a rule and a substitution function that will identify the head of the

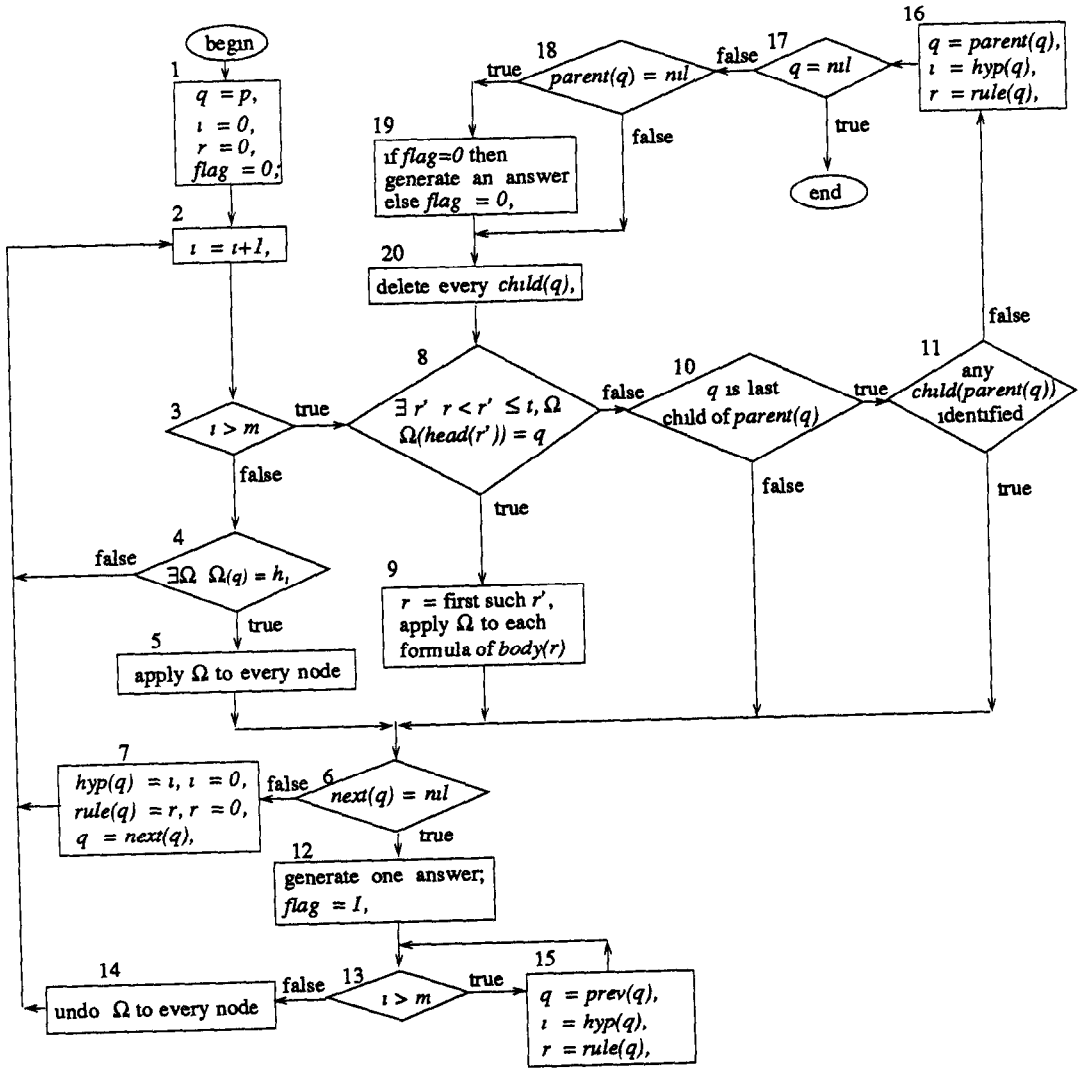

Figure 1 Computing knowledge answers in the non-recursive case

rule with q If such rule and substitution exist (9), we select the first such rule, we apply the substitution to the rule, and add each of the formulas of the body of the rule as a child of q [3] If we cannot find a rule whose head can be identified with q (10), then the process of identifying q fails

If q is the rightmost sibling, and the identification failed for all the siblings, then the rule that added these siblings is no longer productive and we return to the parent formula, restoring its previous state (16) Before discarding this rule, we test whether q is the root (18) If so, then we test the flag if it is unset (this rule was never productive), an answer is generated (19, see below), if it is set, we unset it (for the next rule) In any case, we delete the children of q (20), and attempt to identify q with the head of another rule (8) (The test for termination, 17, will be discussed shortly)

If the identification of q succeeded (either with a formula of a hypothesis or with the head of a rule), or if the identification failed, but q is not the last sibling, or q is the last sibling, but some other sibling was identified, then the entire process is repeated (after the current state of q is saved) with $next(q)$ (6–7)

If q was the last formula of the tree, then an answer is generated (12, see below) and the flag is set (indicating that the rule applied at the root is productive) To generate additional answers, the process is *backtracked*, as follows We look for the most recent formula that was identified (this may be the present formula), restore its state, undo the substitution function that was applied (13–15), and continue with attempts to identify it with the next formula of the hypothesis Note that because of

[3] This function should also rename the other variables of the rule (those in the body but not in the head) so that they do not conflict with currently used variables

the traversal order, the backtracking process visits children before their parent Therefore, before we backtrack to a non-leaf, a formula that identified is always found, and this identification was obviously with a formula of the hypothesis

Eventually, the backtracking process reduces the tree back to a single formula p, and exhausts the different identifications of p It terminates when it tries to backtrack beyond the parent of p (17)

Answers are generated in two situations First, whenever the identification process is completed for the last formula of the tree (12) The answer is then a rule with the root formula for head and the conjunction of the leaf formulas that have not been identified for body (if all leaf formulas have been identified, the rule does not have a body) Second, whenever a rule that was attempted at the root proved unproductive (19) The answer is then a rule with the root formula for head and the conjunction of all the descendent formulas for body

Special care must be taken with formulas that involve built-in predicates (comparisons) Let α and β denote formulas of this kind in the hypothesis and in the tree, respectively Unlike other predicates, even when α and β involve the same predicate, the semantics of the variables could be different, and identification may lead to unintuitive answers Therefore, the algorithm should not identify such formulas Before issuing an answer, every comparison formula of the body (β) is checked against every comparison formula of the hypothesis (α) If the corresponding variables are identical and $\alpha \Rightarrow \beta$, then β is removed from the answer If the corresponding variables are identical and $\neg(\alpha \wedge \beta)$, then this answer is discarded (if all answers have been discarded in this way, the algorithm issues a special answer that indicates that the hypothesis in the query contradicts the IDB)

For reasons of space, the formal proof of the termination of Algorithm 1 and the soundness of its output is omitted

Example 5. When is an honor student eligible for a teaching assistantship in a course currently taught by Susan?

describe $can\_ta(X, Y)$
where $honor(X)$ **and** $teach(susan, Y)$

Algorithm 1 issues this answer

$$can\_ta(X, Y) \leftarrow complete(X, Y, Z, 4\ 0)$$
$$cat\_ta(X, Y) \leftarrow complete(X, Y, Z, U) \wedge (U > 3\ 3) \wedge taught(susan, Y, Z, W)$$

The student must have completed this course with a 4 0, or he must have taken it from Susan with at least 3 3

5 Computing Knowledge Answers in the General Case

Algorithm 1 assumed that the subject of the knowledge query is not recursive (and does not depend on a recursive predicate) First, we examine the various difficulties that are caused when this algorithm is applied to predicates that are recursive (or depend on recursive predicates) Then, we cite a rule transformation that restructures the recursive IDB rules This transformation is then combined with two additional restrictions to create a modified algorithm for general knowledge queries

5.1 The Difficulties

Example 6. When is a course X prior to another course Y, given that databases is prior to Y?

describe $prior(X, Y)$
where $prior(databases, Y)$

Clearly, the subject of this query is recursive If Algorithm 1 were applied to this query, the following infinite answer would be generated

$$prior(X, Y) \leftarrow (X = databases)$$
$$prior(X, Y) \leftarrow prereq(X, databases)$$
$$prior(X, Y) \leftarrow prereq(X, Z_1) \wedge prereq(Z_1, databases)$$

$$prior(X, Y) \leftarrow prereq(X, Z_1) \wedge \quad \wedge\ prereq(Z_{n-1}, Z_n) \wedge prereq(Z_n, databases)$$

Except for the first rule, the body of each rule expresses a "chain" of prerequisites, beginning with X and ending with *databases* Intuitively, the following finite answer expresses the same situation

$$prior(X, Y) \leftarrow (X = databases)$$
$$prior(X, Y) \leftarrow prior(X, databases)$$

Example 7. When is a course X prior to another course Y, given that X is prior to databases?

describe $prior(X, Y)$
where $prior(X, databases)$

Again, the subject of this query is recursive If Algorithm 1 were applied to this query, the following infinite

answer would be generated

$$prior(X,Y) \leftarrow (Y = databases)$$
$$prior(X,Y) \leftarrow prereq(X,X)$$
$$prior(X,Y) \leftarrow prereq(X,Z_1) \wedge prereq(Z_1,X)$$

$$prior(X,Y) \leftarrow prereq(X,Z_1) \wedge \quad \wedge \; prereq(Z_{n-1},Z_n) \wedge prereq(Z_n,X)$$

Except for the first rule, the body of each rule expresses a "loop" of prerequisites, beginning with X and ending with X These rules are not sound They were introduced because the substitution function Ω did not obey the typing requirement

For a final example, consider an EDB with predicates r and s and this simple IDB

$$p(X,Y) \leftarrow q(X,Z) \wedge r(Z,Y)$$
$$q(X,Y) \leftarrow q(X,Z) \wedge s(Z,Y)$$
$$q(X,Y) \leftarrow s(X,Y)$$

Example 8

describe $p(X,Y)$
where $r(a,Y)$

The subject of this query depends on a recursive predicate If Algorithm 1 were applied to this query, it would construct an infinite derivation tree The children of the root would be q and r, after which it will repeatedly add to the left sibling q two new nodes q and s

Altogether, we observe three kinds of problems nontermination that produces an infinite number of answers (Example 6), non-termination that "hangs" over a particular answer (Example 8), and erroneous (and infinite) answers due to type conflicts (Example 7)

5.2 The Transformation

To generate answers similar to the preferred answer shown in Example 6, we rely on results obtained by Imielinski [4] in his research on intensional answers (answers to data queries that include rules) Imielinski defines a rule transformation that serves our purpose here This transformation is defined below

Let p be a recursive predicate of arity n, and let $\Sigma = \{r_1, \quad, r_k\}$ be the recursive rules with head p Let w_i denote the body of the rule r_i without the occurrence of the predicate p Let α_i denote the positions of the variables of the predicate p (either in its occurrence in the head or in its occurrence in the body) that are shared with w_i Let $\alpha = \cup_{r_i \in \Sigma} \alpha_i$ and assume $\alpha = \{i_1, \quad, i_m\}$

Let t be a new predicate symbol, with artity $2m$ (i e , double the cardinality of α) The set of rules Σ is replaced with a set of rules Σ', defined as follows

1 A *transformation* rule, r_T

$$p(U_1, \quad, U_n) \leftarrow p(X_1, \quad, X_n) \wedge t(X_{i_1}, \quad, X_{i_m}, Z_{i_1}, \quad, Z_{i_m})$$

where

$$U_i = \begin{cases} Z_{i_j} & \text{if } i = i_j \in \alpha \\ X_i & \text{otherwise} \end{cases}$$

Thus, some of the variables in the head of r_T (those whose positions are not in α) remain identical to variables in the body formula p The remaining variables in the head of r_T correspond to the Z variables in the body formula t The remaining variables in the body formula p correspond to the X variables in the body formula t

2 For each w_i, an *initialization rule* for t, r_{I_i}

$$t(X_{i_1}, \quad, X_{i_m}, Z_{i_1}, \quad, Z_{i_m}) \leftarrow w_i(A_{j_1}, \quad, A_{j_r}, B_1, \quad, B_s, C_{k_1}, \quad, C_{k_t})$$

where the As are X variables that appear in the body occurrence of p and in w_i, the Bs are variables that appear only in w_i, and the Cs are variables that appear in the head occurrence of p and in w_i The indexes of the As and the Cs preserve the positions of the variables in body and head occurrences of p, respectively Moreover,

$$Z_i = \begin{cases} X_i & \text{if } i \notin \alpha_i \\ C_i & \text{otherwise} \end{cases}$$

3 A *continuation rule*, r_C

$$t(X_1, \quad, X_m, Z_1, \quad, Z_m) \leftarrow t(X_1, \quad, X_m, Y_1, \quad, Y_m) \wedge t(Y_1, \quad, Y_m, Z_1, \quad, Z_m)$$

This transformation is shown in [4] to preserve equivalence (i e , the rules Σ and Σ' extend p in the same way) As an example, the rules

$$prior(X,Y) \leftarrow prereq(X,Y)$$
$$prior(X,Y) \leftarrow prereq(X,Z) \wedge prior(Z,Y)$$

would be transformed to

$$prior(X,Y) \leftarrow prereq(X,Y)$$
$$prior(X,Y) \leftarrow prior(Z,Y) \wedge t(Z,X)$$
$$t(Z,X) \leftarrow prereq(X,Z)$$
$$t(X,Y) \leftarrow t(X,Z) \wedge t(Z,Y)$$

5.3 The Solution

In Section 5 1 we illustrated three problems that occur when Algorithm 1 is used for evaluating recursive queries We begin by considering the problems demonstrated by Examples 6 and 8

Recall that at some point Algorithm 1 finds a rule r that can be identified with the current formula q, and

the tree is then extended with the formulas of the body of r Later, when r ceases to be productive (i e , none of the body formulas can be identified anymore), the algorithm reverses the effect of applying r Consider now the situation when r is recursive There are two troublesome possibilities (1) The above process of applying r and extending the tree may be repeated infinitely (maybe involving some other recursive rules as well), without ever finishing the current derivation tree (Example 8) (2) After r is applied, the descendent formula whose predicate is the same as q's predicate is identified, and an answer is generated Then, in the backtracking process, the same rule r would be applied to that descendent, eventually generating a second answer This process could continue forever, generating an infinite number of answers (Example 6)

Assume now that the transformation described in Section 5 2 is applied to all the recursive predicates in the IDB The structure of the recursive rules after the transformation is such that we can *limit* the number of times that they are applied without "losing" any answers

Consider again the formula q The only recursive rule that can be applied to it is of the kind r_T, and it adds two descendent formulas one with the predicate of q and the other with the artificial predicate t The only recursive rule that can be applied to the latter formula is r_C, and it adds two descendent formulas, both with predicate t The rules r_T and r_C can now be applied again to leaf formulas that have the appropriate predicate We argue that by limiting the subtree of q to at most a single application of r_T and at most two applications of r_C, the answers that would be generated would be "sufficient", that is, any formulas that would be generated by further application of these rules would be redundant The maximal application of recursive rules is shown in Figure 2 The formal proof for this claim is omitted (a similar, though not identical, claim is shown in [5])

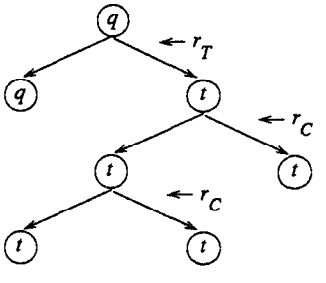

Figure 2 Maximal application of recursive rules

While this correction solves the problems shown in Examples 6 and 8, the problem shown in Example 7 still persists To correct it we must assure that the formulas generated by the algorithm would all obey the same typing requirement that was imposed on all the recursive rules of the IDB In other words, a substitution function that is applied to the entire tree, so that the current tree formula would identify with a hypothesis formula, must not create a situation where the formulas in the tree exhibit a type conflict in a recursive predicate

We shall now show how to modify Algorithm 1 to incorporate these changes

Algorithm 2 Compute knowledge answers in the general case

Input IDB with transformed rules $r_1, \ldots, r_n$, a query with a subject p and qualifier (hypothesis) $\psi = h_1 \wedge \ldots \wedge h_m$

Output: A set of rules $p \leftarrow \varphi$ that are logically derived under the hypothesis ψ

Algorithm 1 is modified to *tag* each formula of the tree A formula is either untagged, or is tagged by the number 0, 1, or 2 The tag 0 prohibits the application of a recursive rule to this formula, other tags (or the absence of a tag) permit the application of a recursive rule

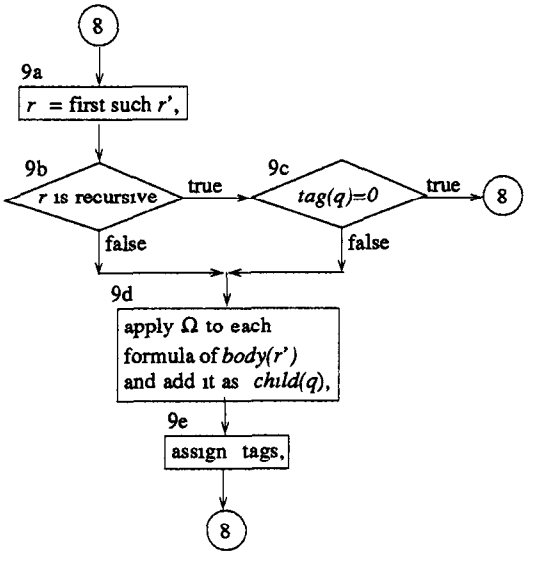

Figure 3 Computing knowledge answers in the general case

First, the substitution functions that identify the current formula with a formula of the hypothesis (box 4) are now required to preserve the types of all the formulas of the tree A substitution function is disqualified, if it will cause two formulas with same predicate to have the same variable in different positions [4]

The flowchart of Algorithm 2 is similar to that of Algorithm 1, except that box 9 is replaced by boxes 9a–9e Figure 3 shows how these five boxes connect to the rest of the flowchart in Figure 1

[4] While this condition is sufficient, it is not necessary, with further computation, more substitution functions could be qualified

When a rule is found (box 9a), we check whether it is recursive (i e, of type r_T or r_C) If not, the algorithm continues as usual (9d) If the rule is recursive, we check the tag of q If the tag is 0, then this rule may not be applied, and another rule is searched (8) Otherwise, the rule is applied (9d) Following the application of the rule, we update the tags of q and its descendents (9e) if the rule was of type r_T, we assign to the descendent with the artificial predicate t the tag 2, and to the other descendent the tag 0, if the rule was of type r_C and q has tag 2, we assign to one of its descendents the tag 1, and to the other the tag 0, if the rule was of type r_C and q has the tag 1, we assign to both of its descendents the tag 0, if the rule is not recursive, no tags are assigned

Again, the formal proof of the termination and soundness of this algorithm is omitted

When Algorithm 2 is applied to Example 6, it generates this answer

$$prior(X,Y) \leftarrow (X = databases)$$
$$prior(X,Y) \leftarrow t(X, databases)$$

This answer is identical to the intuitive answer given earlier, except that t was called *prior*

This example demonstrates the importance of assigning meaningful names to the artificial predicates introduced by the rule transformation of Section 5 2 (answers with mechanically generated predicate names, such as t, tend to have little significance) In reality, it may sometimes be difficult to find intuitive names for these artificial predicates We observe that, under certain circumstances, a slightly modified rule transformation may be used, that avoids a new predicate altogether (and therefore the need to assign a meaningful name) In the previous example, our modified transformation would generate these rules instead

$$prior(X,Y) \leftarrow prereq(X,Y)$$
$$prior(X,Y) \leftarrow prior(X,Z) \wedge prior(Z,Y)$$

And the answer to the same query would then be

$$prior(X,Y) \leftarrow (X = databases)$$
$$prior(X,Y) \leftarrow prior(X, databases)$$

While these answers are equivalent, the second answer is clearly preferable

We have assumed that all recursive rules are typed with respect to their head predicate This restriction may be relaxed with rules of certain structure For example, assume that

$$p(X_1, \quad , X_n) \leftarrow p(X_{i_1}, \quad , X_{i_n})$$
$$p(X_1, \quad , X_n) \leftarrow q(Y_1, \quad , Y_m)$$

where q is not dependent on p, and Ys may be Xs Obviously, the first rule is not typed with respect to its head predicate To rules of this kind we do not apply the transformation, but control the number of times that the recursive rule may be applied This would allow queries similar to the sixth example of the introduction ("When x is reachable from y, is it guaranteed that y is also reachable from x?")

6 Conclusion

Database knowledge is usually applied in the evaluation of queries In this paper we argued that when this knowledge is of substantial volume and complexity, there is genuine need to query this repository of information Moreover, since users of the database may not be able to distinguish between information that is data and information that is knowledge, access to knowledge and data should be provided with a single, coherent instrument

We formalized a framework of knowledge-rich databases with a simple query language consisting of a pair of **retrieve** and **describe** statements The **retrieve** statement is for querying the data (it corresponds to the basic retrieval statement of various knowledge-rich database systems) The **describe** statement is for querying the knowledge Essentially, it inquires about the meaning of a concept under specified circumstances We provided algorithms for evaluating sound and finite knowledge answers to **describe** queries, and we demonstrate them with examples Much work remains to be done, and we mention here several directions that we are currently pursuing

First, issues of redundancy and completeness of knowledge answers require further investigation To a large degree, the algorithms we described for constructing knowledge answers avoid redundancies, by not constructing certain derivation trees Still, some redundancies may go undetected, including redundancies that originate from the IDB rules themselves (e g , when two rules have the same head, but the body of one rule is a consequence of the body of the other) Similarly, in some situations the knowledge answers provided are complete no additional sound and nonredundant formulas exist However, in certain queries, some sound formulas are not generated In this connection we note that the evaluation of the answer to **describe** p **where** ψ is not necessary identical to the answer to **retrieve** p **where** ψ Intuitively, the identification process used for eliminating formulas that are subsumed in the hypothesis from the answer, may sometimes also reduce the generality of the answer

Second, we assumed that the qualifier ψ is a positive formula We are interested in generalizing this formula to allow disjunctions and negations

Third, we are interested in extensions to the present **describe** statement to allow new kinds of queries We sketch here four such extensions Recall, that the semantics of the **describe** statement admitted answers whose body φ and the hypothesis ψ, are together sufficient to derive the subject p Consequently, hypothesis formulas that are entirely unnecessary for the derivation are simply ignored For example, a query to describe the honor students, and a query to describe the honor students that have taken the database course, are answered identically At times, it may be beneficial to provide only those answers where ψ proved necessary For example, the answer to the query

describe $honor(X)$
where necessary $complete(X, Y, Z, U)$ **and** $(V > 3\ 3)$

would generate only those answers were both formulas of the qualifier were necessary A second extension is demonstrated by this query that would inquire whether honor status is *necessary* for teaching assistantship

describe $can\_ta(X, Y)$
where not $honor(X)$

The answer *false* would indicate that honor status is necessary for teaching assistantship A third extension is demonstrated by this subjectless query that would inquire whether students with GPA under 3 5 are *allowed* to be teaching assistants

describe
where $student(X, Y, Z)$ **and** $(Z < 3\ 5)$
and $can\_ta(X, U)$

The answers *true* and *false* would indicate whether this hypothetical situation is possible A fourth extension is demonstrated by this query that would inquire about the advantages of honor status

describe $*$
where $honor(X)$

The "wildcard" subject would express all the subjects that are derivable from this qualifier

Finally, we are interested in knowledge queries that would allow users to *compare* two concepts A possible syntax would combine two **describe** statements

compare
(**describe** p_1
where ψ_1)
with
(**describe** p_2
where ψ_2)

The answer should elucidate the maximal shared concept (if it is empty then the two concepts are *unrelated*, if it is equal to one of the given concepts, then one concept *subsumes* the other)

References

[1] D Chimenti, T O'Hare, R Krishnamurthy, S Naqvi, S Tsur, C West, and C Zaniolo An overview of the LDL system *Database Engineering*, 10(4) 52–62, Dec 1987

[2] L Cholvy and R Demolombe Querying a rule base In *Proc 1st Int Conf on Expert Database Systems* (Charleston, SC, Apr 1–4), pp 365–371, 1986

[3] F Corella Semantic retrieval and levels of abstraction In *Proc 1st Int Workshop on Expert Database Systems* (Kiawah Island, SC, Oct 24–27), pp 91–114, 1984

[4] T Imielinski Intelligent query answering in rule based systems *Journal of Logic Programming*, 4(3) 229–257, Sep 1987

[5] J Minker and J-M Nicolas On recursive axioms in deductive databases *Information Systems*, 8(1) 1–13, 1983

[6] K Morris, J F Naughton, Y Saraiya, J D Ullman, and A Van Gelder YAWN! (yet another window on NAIL!) *Database Engineering*, 10(4) 28–43, Dec 1987

[7] A Motro Using integrity constraints to provide intensional responses to relational queries In *Proc 15th Int Conf on Very Large Data Bases* (Amsterdam, Netherlands, Aug 22–25), pp 237–246, 1989

[8] A Pirotte and D Roelants Constraints for improving the generation of intensional answers in a deductive database In *Proc 5th Int Conf on Data Engineering* (Los Angeles, CA, Feb 6–10), pp 652–659, 1989

[9] K Ramamohanarao, J Shepherd, I Balbin, G Port, L Naish, J Thom, and P Dart The NU-Prolog deductive database system *Database Engineering*, 10(4) 10–19, Dec 1987

[10] R Ramnarayan and H Lu A data/knowledge base management system *Database Engineering*, 10(4) 44–51, Dec 1987

[11] D Sacca, M Dispinzeri, A Mecchia, C Pizzuti, C Del Gracco, and P Naggar The advanced database environment of the KIWI system *Database Engineering*, 10(4) 20–27, Dec 1987

[12] C D Shum and R Muntz Implicit representation for extensional answers In *Proc 2nd Int Conf on Expert Database Systems* (Tysons Corner, VA, Apr 25–27), pp 257–273, 1988

[13] C D Shum and R Muntz An information-theoretic study on aggregate responses In *Proc 14th Int Conf on Very Large Data Bases* (Los Angeles, CA, Aug 25–28), pp 479–490, 1988

[14] J D Ullman *Database and Knowledge-Base Systems, Volume I* Computer Science Press, 1988