MONOTONICALLY IMPROVING APPROXIMATE ANSWERS TO RELATIONAL ALGEBRA QUERIES †

by

Kenneth P. Smith
J. W. S. Liu

1304 W. Springfield Avenue
Department of Computer Science
University of Illinois
Urbana, Illinois 61801

(217)-333-0135
{ksmith, janeliu}@p.cs.uiuc.edu

† This work was partially supported by NASA Contract No. NASA NAG 1 613 and US Navy ONR Contract No. N00014 87-K-0827.

Monotonically Improving Answers to Relational Algebra Queries

ABSTRACT

We present here a query processing method that produces approximate answers to queries posed in standard relational algebra. This method is monotone in the sense that the accuracy of the approximate result improves with the amount of time spent producing the result. This strategy enables us to trade the time to produce the result for the accuracy of the result. An approximate relational model that characterizes approximate relations and a partial order for comparing them is developed. Relational operators which operate on and return approximate relations are defined.

1. INTRODUCTION

In recent years, an increasing amount of attention has been directed toward the design of database systems suited for real–time applications, such as machine vision, multiple robots, avionics, and air traffic control. Such databases can be used to support shared data access by processes in hard real–time systems. In a (hard) real–time system, every real–time process must meet its time constraint, which is typically specified in terms of its deadline. It is essential for the process to complete its execution and produce its result by its deadline. Otherwise, a timing fault is said to occur, and the result produced by the process is of little or no use. Unfortunately, many factors can cause processes to miss their deadlines. Take factors involving a database system for example. Over a period of time, the number of processes and the amount of data accessed by them may vary. Consequently, the number of concurrent queries that are made to the database and the external deadlines by which the queries must be answered may vary. These variations make it difficult to predict the amount of time needed to answer the queries and to schedule the processes requiring the answers to meet all deadlines.

One effective approach to avoidance of timing faults is to trade result accuracy for computation time. Such a tradeoff can be realized by using imprecise computation techniques [1–4]. In this approach, every (hard real–time) process is designed to be *monotone*, that is, the accuracy of its intermediate result improves with the amount of time it executes to produce the result. The final result produced by the process upon its completion is the desired, exact result. Intermediate, approximate results produced by each process at appropriate instances of the process's execution are recorded. If it is necessary to terminate the process prematurely at its deadline, the last approximate result produced is made available. In this way, graceful degradation and scheduling flexibility are achieved.

We refer to algorithms that produce approximate results of increasing accuracy with increasingly longer running time as *monotone* algorithms. Clearly, the imprecise computation approach is feasible only when there are monotone algorithms. While monotone algorithms exist in problem domains such as numerical computation, statistical estimation and prediction, sorting and searching, etc., they do not exist in the query processing domain. The problem of providing monotone query processing algorithms was first addressed in [5,6]. Specifically, Buneman and Davidson present in [6] a query processing algorithm that produces approximate answers of increasing accuracy with increasing query processing time. Their algorithm uses a set of rules written prior to the submission of the query. The rules capture relationships among the data and allow approximate answers to be derived.

In this paper, we present a monotone algorithm that produces improving approximate answers to queries posed in standard relational algebra. Our method differs from Buneman and Davidson's in that the approximate answers are derived directly from the query expression without use of rules. We do this by first developing an approximate relational model. The model defines the approximations of any standard relation, a partial–order relation over the set of all such approximate relations, and the approximate results of each relational operator. We assume here that time for data retrieval, not processing time, contributes to most of the delay in generating any answer. We show how to derive an approximation of the final answer, using whatever data has presently been retrieved. As more data arrive, this approximate answer is improved according to the partial order relation. We also show how rules, such as Buneman and Davidson's, can be easily integrated into our method.

Our work complements the primary/alternate approach [7,8]. This approach to providing approximate answers requires the scheduling of an alternate computation that takes a shorter amount of time to complete when the available time is insufficient to complete the desired primary computations. The processing time required to support such a scheduling decision is often too high to make this approach practical. Our work also complements the recent efforts in design of concurrency control and database management methods suited for real–time applications [9,10].

The remaining part of this paper is organized as follows. In Section 2, we describe our approximate relational model. What approximate answers are and in what sense one answer is better than another are discussed. In Section 3, we define relational algebra operators which operate on and return approximate relations. In Section 4, we present our algorithm for deriving approximate answers to relational algebraic queries. In Section 5, we summarize our results and present our conclusions.

2. AN APPROXIMATE RELATIONAL MODEL

Our definition of an approximate relation is based on the notion of supersets and subsets of tuples in a relation; this idea draws strongly on the work in [6]. Specifically, a relation is a set of tuples. Both supersets and subsets of the relation are useful, meaningful approximations. We know that any tuple not in a superset of a relation certainly does not belong to the relation. On the other hand, all tuples in a subset of a relation certainly belong to the relation. Given a superset and a subset of a relation, they partition the set of all tuples into three blocks with respect to the membership in the relation: tuples certainly in (that is, they are in the subset),

tuples certainly out (that is, they are not in the superset), and tuples whose memberships are uncertain (that is, they are in the difference of the superset and the subset.) We use two of these blocks as the two parts of our approximate relation: the ones which are certainly in and the uncertain ones. A traditional, standard relation is simply one in which the uncertain block is empty.

An *approximate relation* R of a standard relation S is a subset of the cartesian product of a set of domains with the following property: This subset is partitioned into two blocks, C and P. C contains tuples with *certain* membership in the standard relation S. P contains tuples with *uncertain* membership in S. By uncertain membership, we mean that the tuples in P are not known to be in S and are not known to not be in S. These are possible tuples. We denote the approximate relation R by the 2–tuple (C, P). When it is necessary to distinguish the blocks C and P of tuples that characterize different approximate relations R_i, identified by some index i, we denote these sets as C_i and P_i, respectively.

Example 1: Consider a guest list for an office party. You and your friend are putting on a party for some people in your office. You each invite several people. As you prepare to go shopping, you try to prepare a final guest list. You are certain of the people you have invited, but uncertain exactly whom else from your office your friend has invited. The guest list may be represented in an approximate relation R_1 where C_1 contains the guests you invited and P_1 contains the remainder of the office, since all are candidate guests. Figure 1 shows R_1. Your office includes Bill, Joyce, Jorge, Jane, Sue, Jim, Mike, and Fred; and you have personally invited Bill, Joyce, Jorge, and Jane. Figure 2 shows what a standard relation representing the actual guest list approximated by R_1 might be. R_1 approximates the actual guest list, since its certain tuples form a subset of the guest list, and its certain and uncertain tuples together form a superset of the guest list.

For any standard relation S, there is a set of approximate relations that can approximate it. In particular, we say that an approximate relation R_i is an *approximation* of a standard relation S if its subset of certain tuples is a subset of S and R_i itself is a superset of S. In other words, $C_i \subseteq S$, and $C_i \cup P_i \supseteq S$. Given a set $\mathbf{R}$ of approximations of S, there is a partial order relation $\geq$ defined over $\mathbf{R}$. The approximate relation R_i is said to be *better* than or equal to the approximate relation R_j, denoted as $R_i \geq R_j$, if the subset of certain tuples of R_i contains the subset of certain tuples of R_j and the subset of possible tuples of R_i is contained in the subset of possible tuples of R_j. In other words, $C_i \supseteq C_j$ and $P_i \subseteq P_j$. Furthermore, we require that $C_i - C_j$ be a subset of the tuples in P_j. The tuples in $C_i - C_j$ can be thought of as *migrating*

from the uncertain part of R_j into the certain part of R_i as the approximation improves.

Given a standard relation S, let υ be the cartesian product of all domains in the scheme of S. Therefore, υ is the set of all possible tuples which could be in S, or the "universe" of tuples for S. We assume that υ is finite throughout our discussion. Let R_0 is the approximate relation $(\emptyset, \upsilon)$. In other words, R_0 is the approximate relation which represents total uncertainty about S; no tuples are known to be in S or not in S. R_0 is the least, or the worst possible, approximation of S. That is, $R_0 \leq R_i$ for all approximate relations R_i of S. In contrast, S itself is not an approximate relation, but it can be considered as one, $(T_S, \emptyset)$, where T_S is the set of all tuples in S. Clearly, S is the best approximation of S, that is, $S \geq R_i$ for all approximate relations R_i of S.

Consider the partial ordered set $\mathbf{R}$ of approximate relations of a standard relation S and any two approximate relations R_i and R_j in $\mathbf{R}$. An approximate relation $R_i \wedge R_j$ is said to be a greatest common lower bound of R_i and R_j if $R_i \wedge R_j \leq R_i$, $R_i \wedge R_j \leq R_j$, and there is no other approximate relation R_k in $\mathbf{R}$ such that $R_i \wedge R_j < R_k \leq R_i$ and R_j. We note that for any two approximate relations $R_i = (C_i, P_i)$ and $R_j = (C_j, P_j)$, the approximate relation $R_i \wedge R_j = (C_i \cap C_j, P_i \cup P_j)$ is the unique greatest common lower bound of R_i and R_j. Similarly, the approximate relation $R_i \vee R_j = (C_i \cup C_j, P_i \cap P_j)$ is a least common upper bound of R_i and R_j. In other words, $R_i \vee R_j \geq R_i$, $R_i \vee R_j \geq R_j$, and there is no other approximate relation R_k in $\mathbf{R}$ such that $R_i \vee R_j > R_k \geq R_i$ and R_j. In the set $\mathbf{R}$, the relation $R_i \vee R_j$ is the unique least common upper bound of R_i and R_j. Therefore, the partial ordered set $(\mathbf{R}, <)$ of approximate relations of S is a lattice. The relation R_0 defined above is its least element, and the relation S is its greatest element.

Example 2: The approximate relation R_2 in Figure 3 differs from R_1 in Figure 1 in that Sue's tuple is moved from the uncertain part in R_1 to the certain part in R_2. In Figure 4, R_3 has the value of R_1, except that Fred's tuple has been deleted from the uncertain part. Both R_2 and R_3 are approximations of the guest list S in Figure 2. $R_2 \geq R_1$ and $R_3 \geq R_1$. R_2 and R_3 are both better approximations of S than R_1, although they are not mutually comparable by the partial order defined above.

3. RELATIONAL ALGEBRA AND MONOTONICITY

It is not possible to apply standard relational algebra operations to approximate relations. They cannot handle uncertain tuples. We must define new approximate operations, each of which accepts approximate relation(s) as operands and produces an approximate relation as its answer. Furthermore, we are concerned with the quality of the answer produced by any operation. For this reason, we define here the *monotonicity* property of approximate operations. Unlike traditional monotonicity [11], which is concerned with the size of the results, monotonicity is defined here in terms of the goodness of results. Intuitively, an operation is *monotone* if a better set of operands will usually produce a better result, and never a worse one. More formally, monotonicity of an approximate relational algebra operation is defined as follows: Let $\gamma(R_1, \ldots, R_n)$ be such an operation, its arguments and result being approximate relations. Let

$$R_1^1, \ldots, R_n^1 \text{ and } R_1^2, \ldots, R_n^2$$

be two sets of arguments to γ. We say that γ is *monotone* if $R_i^2 \geq R_i^1$, for every i, implies that $\gamma(R_1^2, \ldots, R_n^2) \geq \gamma(R_1^1, \ldots, R_n^1)$.

In the following we define a complete set of monotone relational algebra operations for approximate relations. We describe approximate union, set difference, selection, projection, and cartesian product operators. (Joins can be derived from selection and cartesian product. Intersections can be derived from set difference). For each, we give a proof of monotonicity. Our approximate operations are similar to the operations presented in [12], the differences being that they do not consider monotonicity and we do not consider disjunctive information.

Union

The union of two approximate relations, $R_1 = (C_1, P_1)$ and $R_2 = (C_2, P_2)$, is illustrated by Figure 5. The numbers in Figure 5 represent the 9 cases for each tuple t in υ with respect to the approximate union of R_1 and R_2. They, as listed below, are also used in the examples illustrating other operations.

1) $t \in C_1$ and $t \in C_2$
2) $t \in P_1$ and $t \in C_2$
3) $t \notin R_1$ and $t \in C_2$
4) $t \in C_1$ and $t \in P_2$
5) $t \in P_1$ and $t \in P_2$
6) $t \notin R_1$ and $t \in P_2$
7) $t \in C_1$ and $t \notin R_2$
8) $t \in P_1$ and $t \notin R_2$
9) $t \notin R_1$ and $t \notin R_2$

The approximate union, $R_T = R_1 \cup R_2$, is defined as follows:

$$C_T = C_1 \cup C_2$$
$$P_T = (P_1 \cup P_2) - C_T$$

where "$\cup$" and "$-$" in the right hand side of the equations denote standard set union and standard set difference, not the approximate operations being defined here. As can be seen in Figure 5, C_T consists of the tuples in cases 1, 2, 3, 4, and 7 listed above. P_T consists of the tuples in cases 5, 6, and 8.

We want to show that the union operation is monotone, that is, the better the operands are, the better the result will be. To do so, let R_1, R_1', R_2, R_2', R_T, and R_T' be approximate relations. Also, let $R_1' \geq R_1$ and $R_2' \geq R_2$. $R_T = R_1 \cup R_2$ and $R_T' = R_1' \cup R_2'$. We show below that $R_T' \geq R_T$ by showing that (1) $C_T' \supseteq C_T$ and (2) $P_T' \subseteq P_T$.

To show that $C_T' \supseteq C_T$, we consider an arbitrary tuple t in C_T. By definition, t is in C_1 or C_2. Since $R_1' \geq R_1$ and $R_2' \geq R_2$, we know that $C_1' \supseteq C_1$ and $C_2' \supseteq C_2$. Therefore t is also in C_1' or C_2', and $t \in C_T'$. In other words, an arbitrary tuple in C_T is always also in C_T', and hence $C_T' \supseteq C_T$. Similarly, to show that $P_T' \subseteq P_T$, we consider an arbitrary tuple t in P_T'. $t \in P_1' \cup P_2'$, and $t \notin C_T'$ by definition. Since $R_1' \geq R_1$ and $R_2' \geq R_2$, $P_1' \subseteq P_1$ and $P_2' \subseteq P_2$. Therefore $t \in P_1$ or $t \in P_2$. We have already shown that $C_T' \supseteq C_T$. If follows that $t \notin C_T$. Therefore, $t \in (P_1 \cup P_2) - C_T$, which is P_T. Since an arbitrary tuple in P_T' is always also in P_T, we know that $P_T' \subseteq P_T$.

Set Difference

The difference of two approximate relations is illustrated by Figure 6. The approximate set difference, $R_T = R_1 - R_2$, is defined as follows:

$$C_T = C_1 - R_2$$
$$P_T = (P_1 - R_2) \cup (P_2 \cap R_1)$$

where "$\cup$", "$-$", and "$\cap$" in the right hand side of the equations denote standard set union, set difference, and set intersection. As can be seen in Figure 6, C_T consists of the tuples in case 7 listed above. P_T consists of the tuples in cases 4, 5, and 8.

To show that the difference operator defined here is monotone, we let R_1, R_1', R_2, R_2', R_T, and R_T' be approximate relations and $R_1' \geq R_1$ and $R_2' \geq R_2$. $R_T = R_1 \cup R_2$ and $R_T' = R_1' \cup R_2'$. Again, we want to show that $R_T' \geq R_T$ by showing that (1) $C_T' \supseteq C_T$ and (2) $P_T' \subseteq P_T$. To begin, we prove an intermediate result: If $R_i \geq R_j$, then $C_i \cup P_i \subseteq C_j \cup P_j$. In other words, $R_i \subseteq R_j$. To show that this is true, we note that since $P_i \subseteq P_j$, there are no tuples

in P_i that are not also in P_j. Moreover, $C_i - C_j$ is a set of (migrated) tuples in P_j. Therefore, although $C_i \supseteq C_j$, there is no tuple in C_i that is not also in P_j or C_j.

To show that $C_T' \supseteq C_T$, let t be an arbitrary tuple in C_T. $t \in C_1$, and $t \notin R_2$ by definition of R_T. Since $R_1' \geq R_1$ and therefore $C_1' \supseteq C_1$, $t \in C_1'$. By the result above, if $t \notin R_2$, then $t \notin R_2'$, since $R_2' \geq R_2$. It follows that $t \in C_1' - R_2'$, which is C_T'. Since any arbitrary tuple in C_T is also in C_T', we have $C_T' \supseteq C_T$. Similarly, if t is an arbitrary tuple in P_T', by definition of P_T', either $t \in P_1 - R_2$, or $t \in P_2 \cap R_1$. If $t \in P_1 - R_2$, we wish to show that $t \in P_1' - R_2'$ also. We know that $P_1' \subseteq P_1$ and $R_2' \supseteq R_2$ by the result above. Therefore, $t \in P_1' - R_2'$. Similarly, if $t \in P_2 \cap R_1$, it must also be in $P_2' \cap R_1'$. Therefore, $t \in (P_1 - R_2) \cup (P_2 \cap R_1)$, which is P_T. Hence, we have $P_T' \subseteq P_T$.

Selection

The selection of some value for an attribute of an approximate relation is illustrated by Figure 7. The approximate selection, $R_T = \sigma_{val-att}\ R_1$, is defined as follows:

$$C_T = \sigma_{val-att}\ C_1$$
$$P_T = \sigma_{val-att}\ P_1$$

where $\sigma_{val-att}$ denotes the traditional selection operator that treats C_1 and P_1 as traditional, exact relations. To show that the selection operation is monotone, let R_1, R_1', R_T, and R_T' be approximate relations. Let $R_1' \geq R_1$, $R_T = \sigma_{val-att}\ R_1$, and $R_T' = \sigma_{val-att}\ R_1'$ for some value 'val' and attribute 'att'. That $R_T' \geq R_T$ follows directly from $C_1' \supseteq C_1$, $P_1' \subseteq P_1$, and the definition of the approximate selection.

Projection

The approximate projection, $R_T = \pi_{att}\ R_1$, is defined as follows:

$$C_T = \pi_{att}\ C_1$$
$$P_T = \pi_{att}\ P_1$$

where π_{att} denotes the traditional projection operator operating on C_1 and P_1 as if these subsets were exact relations. Since tuples (certain and uncertain) in R_1 and R_T are in 1–1 correspondence under the definition of the projection operation, any improvement to R_1 will be directly reflected by an improvement in R_T.

Cartesian Product

The approximate cartesian product, $R_T = R_1 \times R_2$, is defined as follows:

$$C_T = C_1 \times C_2$$

$$P_T = (R_1 \times R_2) - C_T$$

where "$\times$" denotes the traditional cartesian product of sets. To show that the approximate cartesian product is a monotone operator, we consider three pairs of approximate relations: $R_1, R_1', R_2, R_2', R_T$, and R_T'. Again, let $R_1' \geq R_1$ and $R_2' \geq R_2$. $R_T = R_1 \times R_2$, and $R_T' = R_1' \times R_2'$. We want to show that $R_T' \geq R_T$. We note that C_T' has four parts: $(C_1 \times C_2)$, $(C_1' - C_1) \times C_2$, $C_1 \times (C_2' - C_2)$, and $(C_1' - C_1) \times (C_2' - C_2)$. Since the first part by itself equals C_T, $C_T' \supseteq C_T$. From the result proved for the set difference operation, $R_1' \subseteq R_1$, and $R_2' \subseteq R_2$. Therefore, $(R_1' \times R_2') \subseteq (R_1 \times R_2)$, because $\times$ is monotonic in the traditional sense. Since $C_T' \supseteq C_T$, a tuple in P_T' (that is, in $(R_1' \times R_2') - C_T'$) is also in P_T. So, $P_T' \subseteq P_T$, and $R_T' \geq R_T$.

4. APPROXIMATE QUERY ANSWERING

In this section we describe how to process relational algebra queries so that the answer improves monotonically as additional information becomes available. To explain our algorithm, we first note that any relational algebra query can be represented as a tree. Each leaf node represents a base relation read by the query processor from the database, it being the operand of some relational algebra operation. Each non–leaf node in the tree represents the result of a relational algebra operation, its child(ren) being the operand(s). The root node represents the final result of the query. For example, the query

$$R_T = \sigma_{att=val} R_x \cup (R_y \times R_z)$$

can be represented by the tree shown in Figure 8. The operation marked below each node is the operation by which that node's value is calculated.

Query answers are typically derived in the following manner: First, each leaf node is evaluated by issuing a read request to the database for the base relation it represents. When all operands represented by its children are available, a node at the next higher level is evaluated. This process repeats until the root node, the query answer, is evaluated. The value given to the root node at the end of the query evaluation process is the exact answer to the query. No intermediate, approximate answer is produced however; no answer for the query exists until every node below the root is fully evaluated. In particular, no answer can be produced until all read requests are granted. If any base relation is not immediately accessible, no answer is available until it becomes accessible. If any base relation is totally inaccessible, no answer is ever produced. As discussed earlier, this all–or–nothing query processing strategy does not degrade

gracefully.

Our query processing strategy differs from the traditional strategy in an important aspect: a series of approximate answers are produced, each integrating the effect of additional base relation data. None but the final, exact answer requires all base relation data be available before it can be produced. In terms of our approximate relational model, the value of the root is set to a series of approximations $R_0, \ldots, R_n$ of the final exact answer A to the query. The first answer $R_0 = (\emptyset, \upsilon)$ is the least approximation of A. υ is the set of all tuples that might be in A The last answer and best approximation R_n is the exact answer A obtained by the traditional algorithm. These two answers, plus intermediate answers in the series are a chain, that is, $R_0 \leq R_1 \leq \cdots \leq R_{n-1} \leq R_n$ in the lattice of approximations of A. If query processing is prematurely terminated (due to a deadline, for instance,) some approximate answer R_k in the chain will be produced, where k increases monotonically with time.

The algorithm starts with the query represented as a tree, as would be provided by a parser. In this tree, all operations are approximate ones as defined in Section 3. Each node of the tree has a value, which is initially set to R_0 for all nodes. At any time, if query processing terminates, the value of the root node is returned by the query processor as the answer.

After read requests begin to be answered, the leaf nodes begin to assume their exact values as read from the database. When the read request of a leaf node N is answered, the effect of the new value of N is *propagated* upward to the root as follows: The value returned by the read operation is interpreted as an approximate relation with no uncertainty, that is, no tuples in the P portion. This value and the value of the sibling of N, if N has a sibling, are taken as the operands of the approximate operation whose result is represented by their parent node N'. The resulting approximate relation is the new value of N'. This improved value is propagated upward in the tree in like manner until the value of the root node is updated and improved.

Let us consider the example in Figure 8 once more. If R_2 is returned from the database, this value is interpreted as an approximate relation and represented by node 1. The approximate selection operator is applied to obtain a new value for node 4. Finally, the approximate union of node 4 and node 5 is evaluated to obtain the new value of the root node, node 6. We are sure that this newer value of node 6 is better than the older one because (1) the value returned from the database for node 1 is better than the initial value R_0 and (2) approximate selection and approximate union are monotonic. Again, monotonicity is defined here in terms of the quality of results.

We can improve this basic algorithm, taking advantage of the expressiveness of the approximate relational model. Two ways are described below:

Better answers can be obtained sooner by making use of partial information from the database. Relations are sometimes horizontally partitioned across a file system that may be distributed across a network. Instead of requiring each read request to return the entire base relation, we may allow *approximate reads*. Each approximate read returns a fragment of the requested base relation if the entire relation is inaccessible at the moment. The returned fragment can be interpreted as an approximate relation and used to improve the value of the root node. As the value of the leaf node improves, due to the arrival of more fragments, the value of the root can be improved by the propagation mechanism just described. In general, any approximation better than the current value of a leaf node can be used to improve the current value at the root node. "Smart" query processors might search the database for ways to construct better approximations of the leaf nodes from data available elsewhere in the database if the base relations are inaccessible. In the semantic data model, information in the schema can designate the subset and superset relationship between data. If a relation known to be a subset or superset were available in lieu of the requested relation, it could be used as an approximation of the leaf node.

In addition to the basic propagation mechanism which starts at leaf nodes, we can make use of partial information to improve the value of non–leaf nodes in the same manner. If a set of available tuples is known to be a superset of node 4 in Figure 8, for example, it can be used to improve the value of node 4. The improvement can be propagated upward as usual to the root node. The rules of [6] implement this in the case of the root node.

If non–leaf nodes are updated directly as described above rather than by propagation beginning at the leaf nodes, one principle must be observed to preserve monotonicity. This principle is that upward propagation must stop if the new value is not better than the current value of a node. For instance, if node 4 has been directly updated, an improvement in node 1 should not be propagated to it unless the result of the approximate selection using the improved value of node 1 as operand is better than the current value of node 4.

5. SUMMARY

We have developed a method for producing approximate answers to relational algebraic queries. This method is based on a set of new relational algebra operators each of which accepts

approximate relations as its input and produces an approximate relation as its result. For each operation, we show that improving the operands will never degrade the result. Thus, an improvement in the operands of an expression containing these operators as primitives will lead to an improvement in the result of the expression. Our method can be viewed as a way of integrating approximate information from the database into the answer, by propagating improved information through the operators in the query expression to reduce the uncertainty in the result. The rules of Buneman and Davidson's approach [6] provide a method of integrating approximate information to form the answer. Their method is a special case of ours in which the answer is improved directly, without employing the relational operators in the query expression.

Our method has the advantage that very small changes from the traditional database architecture are required to implement it. In particular, no change to the underlying database is required, only the interpretation of data by the query processor and the interpretation of answers by the application need be changed. Our approximate relational model gracefully extends the traditional relational model and accounts for uncertainty. When there is no uncertainty, the models are identical. Propagation of improvements leads to higher query processing overhead than the traditonal method. The evaluation of this overhead and ways to reduce it are subjects of our current investigation.

References

[1] Lin, K. J., S. Natarajan, J. W.–S. Liu, and T. Krauskopf, Concord: a system of imprecise computations, *Proceedings of the 1987 IEEE Compsac*, Japan, October, 1987.

[2] Liu, J. W. S., K. J. Lin, and S. Natarajan, Scheduling real–time, periodic jobs using imprecise results, *Proceedings of the IEEE Real–Time Systems Symposium*, San Jose, California, December 1987.

[3] Chung, J. Y., J. W. S. Liu, and K. J. Lin, Scheduling periodic jobs that allow imprecise results, to appear in *IEEE Transactions on Computers.*

[4] Chung, J. Y. and J. W. S. Liu, Performance of algorithms for scheduling periodic jobs to minimize average error, to appear in *Proceedings of the IEEE 9th Real–Time Systems Symposium*, Huntsville, Alabama, December 1988.

[5] Davidson, S. B., and A. Watters, Partial Computation in Real–Time Database Systems, *Proceedings of The Fifth Workshop on Real–Time Software and Operating Systems*, pp. 117–121, May 1988.

[6] Buneman, P., S. Davidson, and A. Watters, A Semantics for Complex Objects and Approximate Queries, *Proceedings of The Seventh Symposium on the Principles of Database Systems*, pp. 305–314, March 1988.

[7] Liestman, A. L. and R. H. Campbell, A fault–tolerant scheduling problem, *IEEE Transactions on Software Engineering*, vol. SE–12, No. 10., pp. 1089–1095, Oct. 1986.

[8] Lesser, V., J. Pavlin, and E. Durfee, Approximate Processing in Real–Time Problem Solving, *AI Magazine*, Vol. 9, Number 1, Spring 1988 pp. 49–61.

[9] Vrbsky, S., and K. J. Lin, Recovering Imprecise Transactions with Real–Time Constraints, *Proceedings: The Seventh Symposium on Reliable Distributed Systems*, pp. 185–193, October 1988.

[10] R. Abbott and H. Garcia–Molina, What is a real–time database system?, *Fourth Workshop on Real–Time Software and Operating Systems*, pp. 134–138, July 1987.

[11] Ullman, J., *Principles of Database and Knowledge Based Systems*, Vol. 1, pp. 121–122 Computer Science Press, Rockville, Maryland, 1988.

[12] Liu, K. and R. Sunderraman, On Representing Indefinite and Maybe Information In Relational Databases, *Proceedings: The Fourth International Conference on Data Engineering*, pp. 250–257, February 1988.

Guest List	
Name	Phone No.
Bill	333–8100
Joyce	333–8100
Jorge	371–2302
Jane	325–2971
Sue	254–2663
Jim	327–1342
Mike	254–0048
Fred	578–9976

Figure 1: An example: an approximate relation R_1

Guest List	
Name	Phone No.
Bill	333–8100
Joyce	333–8100
Jorge	371–2302
Jane	325–2971
Sue	254–2663

Figure 2: An example: the actual guest list

Guest List	
Name	Phone No.
Bill	333–8100
Joyce	333–8100
Jorge	371–2302
Jane	325–2971
Sue	254–2663
Jim	327–1342
Mike	254–0048
Fred	578–9976

Figure 3: Approximate relation R_2

Guest List	
Name	Phone No.
Bill	333–8100
Joyce	333–8100
Jorge	371–2302
Jane	325–2971
Sue	254–2663
Jim	327–1342
Mike	254–0048

Figure 4: Approximate relation R_3

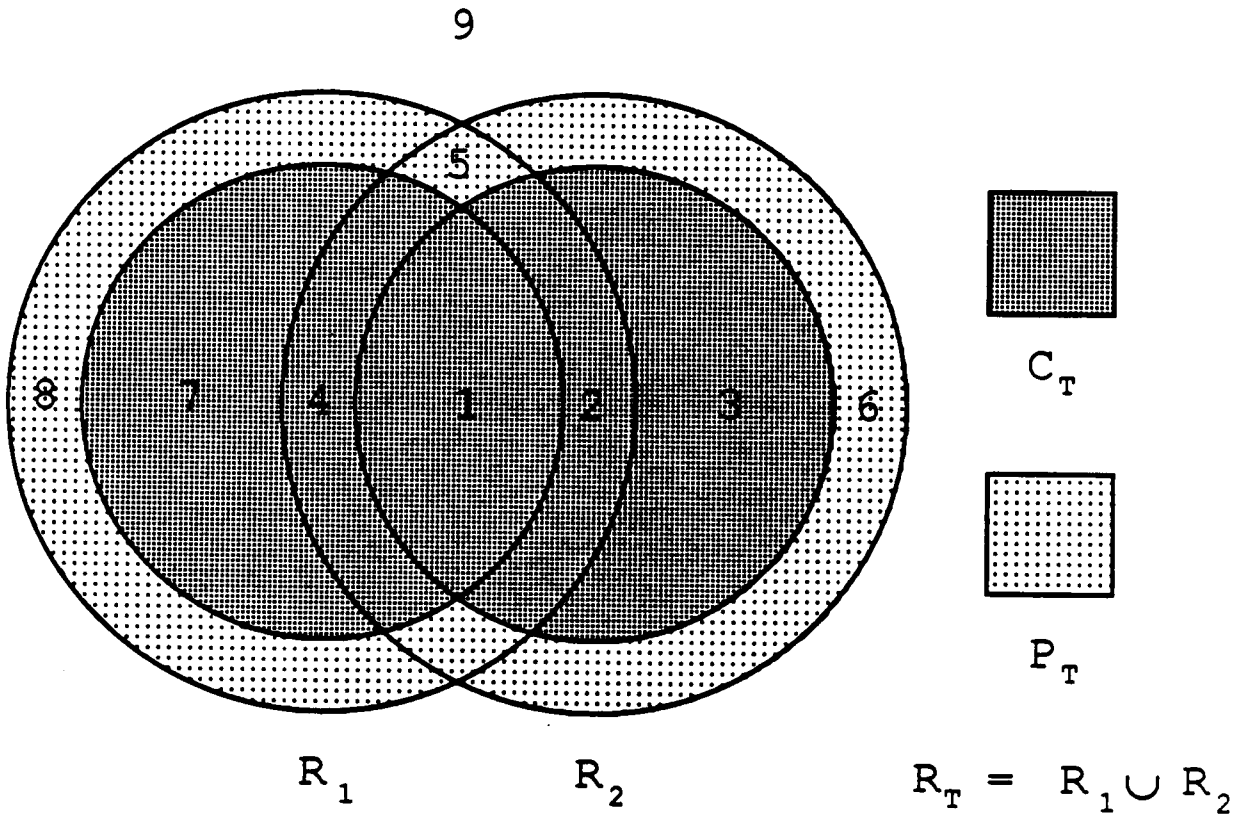

Figure 5. Union of R_1 and R_2

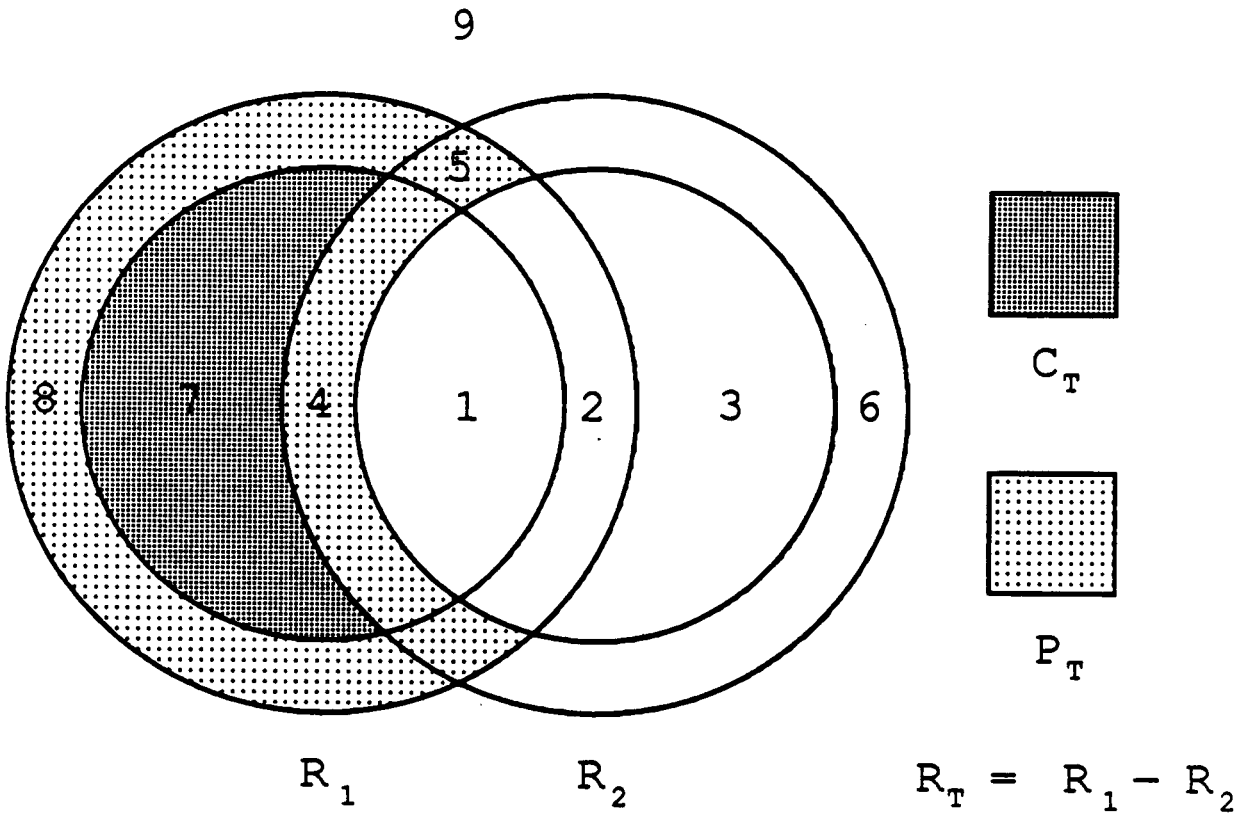

Figure 6. Set difference of R_1 and R_2

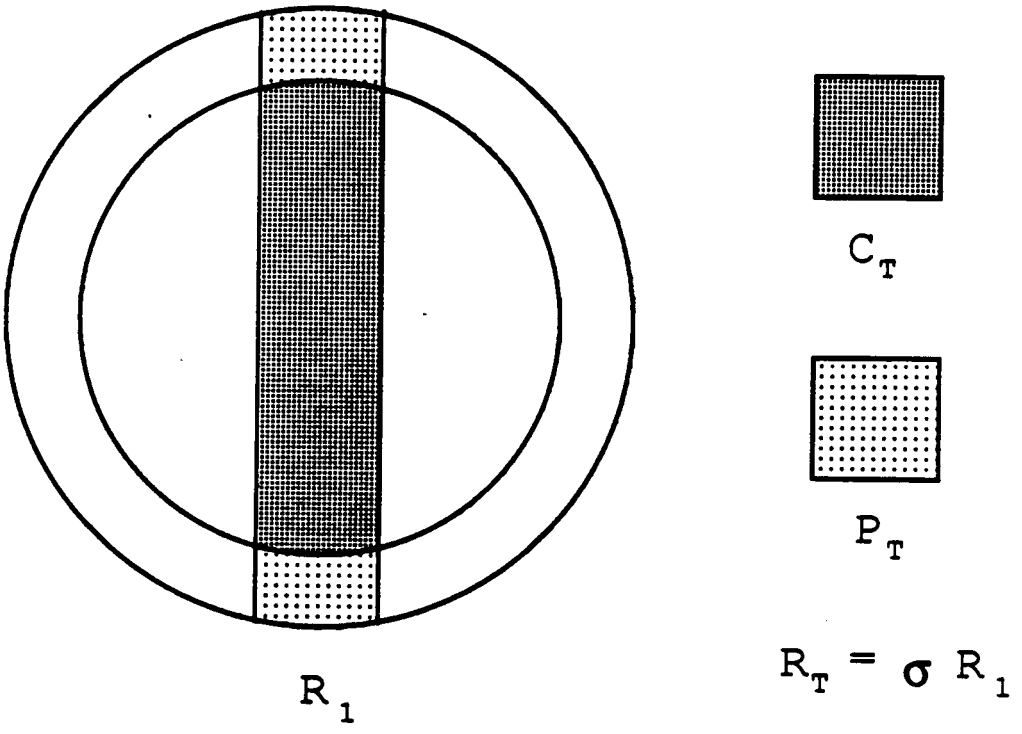

Figure 7. Selection from R_1

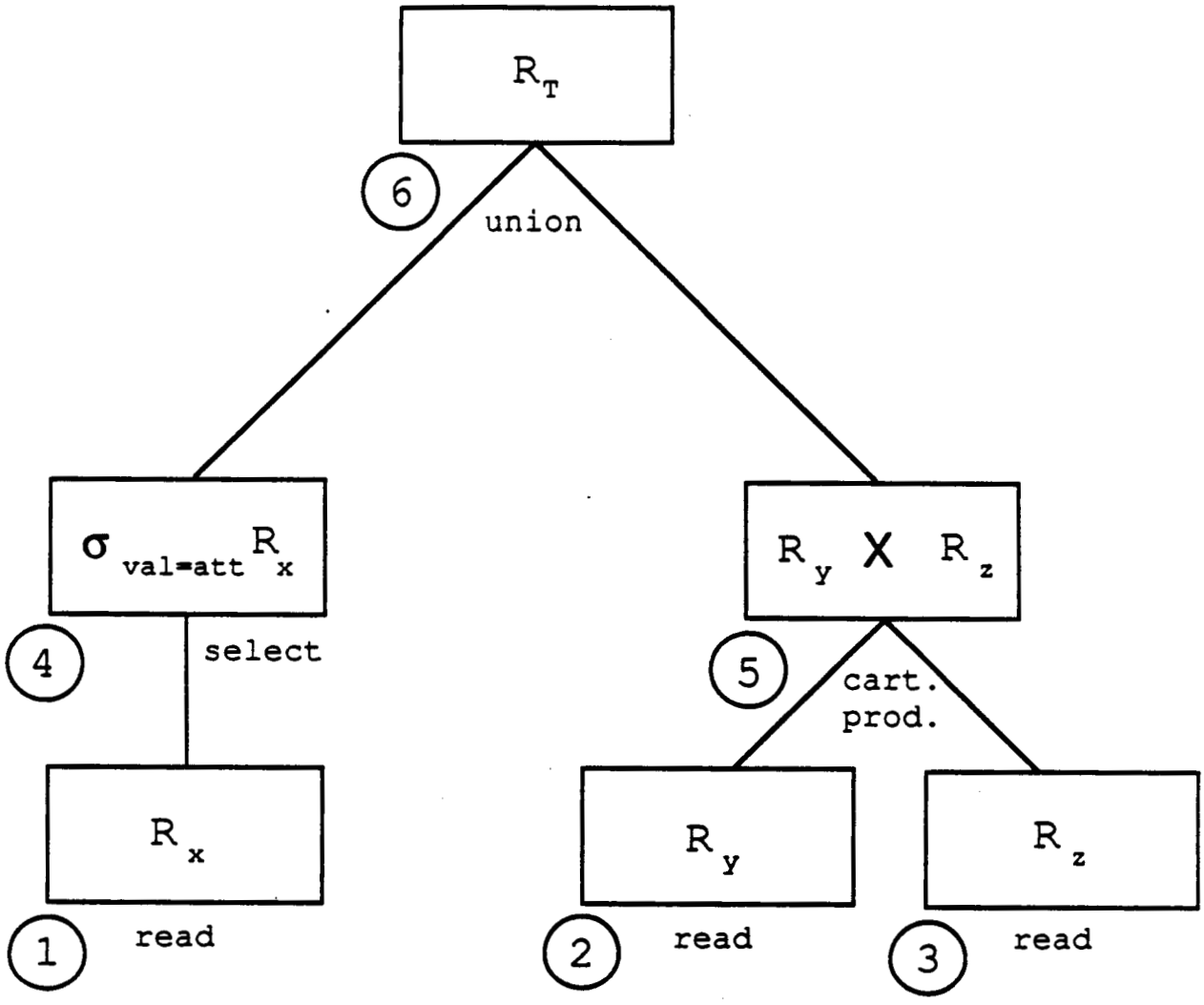

Figure 8. The tree representation of a query