Optimization Using Tuple Subsumption

Venky Harinarayan[1] and Ashish Gupta[2]

[1] Department of Computer Science, Stanford University, CA 94305-2140
(venky@cs.stanford.edu)
[2] Department of Computer Science, Stanford. *and* IBM Almaden Research Center

Abstract. A tuple t_1 of relation R *subsumes* tuple t_2 of R, with respect to a query Q if for every database, tuple t_1 derives all, and possibly more, answers to query Q than derived by tuple t_2. Therefore, the subsumed tuple t_2 can be ignored with respect to Q in the presence of tuple t_1 in relation R. This property finds use in a large number of problems. For instance: during query optimization subsumed tuples can be ignored thereby avoiding the computation of redundant answers; the size of cached information in distributed and object oriented systems can be reduced by omitting subsumed tuples; constraints need not be checked and rules need not be recomputed when provably subsumed updates are made. We give algorithms for deciding efficiently when a tuple subsumes another tuple for queries that use arbitrary mathematical functions. We characterize queries for which, whenever a set of tuples $\mathcal{T}$ subsumes a tuple t then one of the tuples in $\mathcal{T}$ also subsumed t, yielding efficiently verifiable cases of subsumption.

1 Introduction

We discuss and formalize the property of *subsumption* in this paper. Subsumption, intuitively, is the identification of the tuples of a database that do not "contribute" to the result of a query. Subsumption is a powerful optimization technique for a variety of problems. In the introduction we give some examples to illustrate and motivate the use of subsumption. The first example illustrates the use of subsumption to optimize integrity constraint checking.

Example 1. This example is from [GW93]. Consider an employee-department relational database with two relations:

$\texttt{EMP}(E,\ D,\ S)$ *% employee number E in department D has salary S*
$\texttt{DEPT}(D,\ MS)$ *% some manager in department D has salary MS*

Consider a constraint on the database which asserts that every employee earns less than every manager in the same department. This constraint is expressed as a conjunctive query C [Ull89] such that if C derives **panic** the constraint is violated:

$$C\colon\quad \mathbf{panic} :\!-\ \texttt{emp}(E,D,S)\ \&\ \texttt{dept}(D,MS)\ \&\ S \geq MS.$$

We use upper case letters to refer to the relation corresponding to a particular predicate. For instance, EMP refers to the relation corresponding to predicate emp.

Suppose tuple $\texttt{emp}(e1, d1, 50)$ is inserted into relation EMP. Constraint C will be violated if department $d1$ has a manager whose salary is ≤ 50. However, suppose department $d1$ already has an employee whose salary is 100. Since constraint C is not violated before the insertion, we can infer that no manager in $d1$ earns as little as 100, and therefore $\texttt{emp}(e1, d1, 50)$ does not violate constraint C. We say tuple $\texttt{emp}(e2, d1, 100)$ subsumes tuple $\texttt{emp}(e1, d1, 50)$ with respect to constraint query C.

Consider a scenario where the relation DEPT is expensive to access or not accessible at all. Let tuple μ be inserted into EMP. If some existing tuple in EMP subsumes μ then constraint C can be checked using only relation EMP without accessing DEPT.

[GW93, GSUW94] build a theory that uses subsumption to verify integrity constraints that are expressable as Select-Project-Join statements that use arithmetic inequalities. The techniques developed there often lead to more efficient constraint checking than naive strategies that do not exploit subsumption. This is especially true in distributed systems and heterogeneous systems where some relations may be very expensive or impossible to access. However, the results in those papers do not extend to queries that use arbitrary mathematical expressions. In addition, in this paper we characterize subsumption in a noncomputational way that yields insight into the problem and opens new avenues for identifying classes of constraints and updates for which subsumption holds. Now we illustrate other applications of subsumption.

Example 2. Consider a distributed database where the relation EMP is on site 1 and DEPT is on site 2. Let site 2 use view bad_dept defined as follows:

$$C: \quad \texttt{bad\_dept}(D) \coloneq \texttt{emp}(E, D, S) \ \& \ \texttt{dept}(D, MS) \ \& \ S \geq MS.$$

Let site 2 cache the relation EMP for answering queries on view bad_dept. If relation EMP has tuples $\texttt{emp}(e1, d1, 100)$ and $\texttt{emp}(e2, d1, 50)$ then both tuples need not be cached in order to answer queries on view bad_dept. In particular $\texttt{emp}(e2, d1, 50)$ can be dropped from the cache. In general, all subsumed tuples can be dropped from the cache resulting in a smaller cached relation. The tuples that are not subsumed by any other tuple constitute the *representative* relation. The representative relation for emp will be referred to as $\texttt{emp}^{\texttt{r}}$. Additionally, if relation EMP is updated on site 1 and if a subsumed tuple is inserted or deleted, then the cache on site 2 need not be updated!

Example 3. Consider view bad_dept defined in Example 2. Note, the view has the same body as the constraint in Example 1 except that the head has arity 1 and not 0. Just as in Example 1, it is straightforward to observe that tuple $\texttt{emp}(e1, d1, 100)$ subsumes tuple $\texttt{emp}(e2, d1, 50)$ with respect to view bad_dept. Hence, if tuple $\texttt{emp}(e2, d1, 50)$ is inserted into relation EMP and if the relation has a subsuming tuple, then the view update decision can be made without accesing relation DEPT.

Finally, tuple subsumption can be used in query optimization. The notion of subsumption can be pushed into a query at optimization time. Hence, each relation can be reduced to its representative relation before the relation participates in a join. Techniques currently used, like pushing selections down the query tree and the use of semi-join algorithms are instances of this general concept of subsumption. We can do more: in Example 3 the property of subsumption can be used to replace relation $\texttt{EMP}(E, D, S)$ by

Define $\texttt{emp}^{\texttt{r}}$ as **select** $D, \mathbf{max}(S)$ **from** $\texttt{emp}$ **groupby**(D) .

and to replace relation $\texttt{DEPT}(D, MS)$ by

Define $\texttt{dept}^{\texttt{r}}$ as **select** $D, \mathbf{min}(MS)$ **from** $\texttt{dept}$ **groupby**(D) .

The representative relations can be used instead of the original relations to compute `bad_dept`. Note, the representative relations $\texttt{emp}^{\texttt{r}}$ and $\texttt{dept}^{\texttt{r}}$ need not be explicitly defined or materialized; rather the aggregation can be pushed into the original query yielding an alternative query plan for the optimizer to choose from. Thus, subsumption based query rewriting may *introduce* aggregate predicates in a query that originally did not use aggregation. This rewrite can be done automatically using the theory of subsumption. It has been shown in [CS94] that pushing aggregate subgoals down a query tree often results in more efficient plans than the original plan. Thus, we have reason to believe that introducing aggregate subgoals can result in significant savings; often reducing a $O(n^2)$ time query to an $O(n)$ query.

We formally define subsumption later in the paper. For now it is sufficient to define it as follows: consider a query Q on a database consisting of a known set of tuples $\{t_i\}$ in relation R and some unknown relations $\bar{S}$. Now let us augment the relation R with a tuple t. If the answer to query Q does not change, independent of the values of the relations $\bar{S}$, we say that tuple t is subsumed by the set of tuples $\{t_i\}$ with respect to query Q.

STS Property Consider a query Q and tuple t such that the following property holds: t is subsumed by a set of tuples $\{t_i, 1 \leq i \leq n\}$ if and only if t is subsumed by one of the tuples $t_j, 1 \leq j \leq n$ in the set. For such cases we say that single tuple subsumption is the *strongest* possible check and that the tuple t has the STS property with respect to query Q. Or in the language of [GSUW94] we say that single tuple subsumption is the "complete local check"[3]. In the examples given earlier all the tuples had the STS property with respect to the queries considered. The following example is an instance where the STS property does not hold. In such cases we say that multiple tuple subsumption (MTS) holds.

MTS Property Consider a database that stores points (on the number line) in relation `P` and line segments in relation `L`. Let view `free_point` contain those points that are not part of any line segment. Let a line segment $l1$ be inserted into relation `L`. If $l1$ is a subsegment of some other segment $l2$ that is already in relation `L` then we can infer that $l2$ subsumes $l1$ with respect to view `free_point` and that the view stays unchanged. We can make the same inference if $l1$ is a subsegment of (is subsumed by) the union of *two* line segments $l2$ and $l3$. In

[3] *Completeness* is formally defined in [GSUW94]

general, $l1$ could be a subsegment of the union of an arbitrary number of existing lines segments without being a subsegment of any one of them. Note, single tuple subsumption is a degenerate case of MTS and is always a *sufficient* check though possibly not the strongest (*complete*) check. The STS property refers to cases where single tuple subsumption *is* the complete check, *i.e.*, the "if and only if" condition stated before holds.

Inferring subsumption by multiple tuples is computationally more expensive than inferring subsumption by a single tuple: checking for MTS may be exponentially more expensive than checking for STS. More importantly, the MTS check may not be expressable in the query language of the database. For instance, for the point-line example above, the general MTS check may use an arbitrary number of tuples and thus checking MTS may involve iterating/recursing over the entire database. Such a check cannot be expressed in a first order language and thus cannot be written as a SQL query (unless some recursive or iterative construct is used). On the other hand, a single tuple check (be it just sufficient or complete) can be written as a single SQL query. We believe STS will lead to efficient optimizations for the applications given earlier.

Thus it is useful to identify classes of queries for which the STS property holds and can be evaluated efficiently.

We consider integrity constraints, 0-ary queries and for these queries we characterize STS in terms of the existence of a "distinguished point". This characterization is a powerful result as it reduces the problem of determining if a k-dimensional unsafe region is contained in a union of m unsafe regions to the problem of determining if a *point* in k-dimensions is contained in one of m k-dimensional regions.

The above characterization enables us to identify classes of queries that use arbitrary mathematical functions as subgoals and for which it is computationally feasible to check STS. The existential characterization holds for a larger class of constraints than those for which we can compute the distinguished point. [GSUW94] considered the problem of checking the STS property for Select-Project-Join statements that use arithmetic comparison operators $<, \leq, >, \geq, =$. The results in [GSUW94] are an example of the more powerful abstraction of the distinguished point, introduced in this paper. Also [GSUW94] considered the restricted case where the STS property holds for a given constraint but does not consider constraints where the STS property may hold for a constraint and a particular tuple but not for the same constraint and some other tuple.

The results of this paper enable subsumption to be checked for queries that use functions like distance, volume, and other more complicated mathematical functions. Such functions appear often in constraint and query specifications [TH93]. Therefore, we need a general theory that deals with subsumption in the presence of arbitrary arithmetic constraints. For instance, consider the following example:

Example 4. Consider a factory floor with a robotic arm that is hinged at one end and rotates in a two dimensional plane. The free end of the arm defines a circular locus. However, the arm does not have 360^o rotational freedom. The

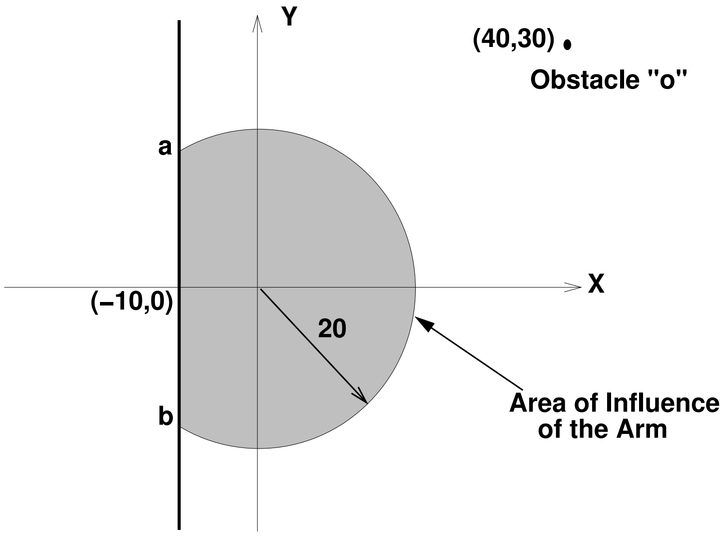

Fig. 1. Figure for Example 4

shaded area in Figure 1 shows the area of influence of the arm.

The factory also has obstacles, like machines, that interfere with the robot's arm if the obstacle is placed close enough to the arm such that some point in the obstacle lies in the shaded area for the arm. We will represent these obstacles as points in the two dimensional plane of motion of the robot arm.

One of the constraints in the above scenario is that an obstacle should not be placed in a position where it can interfere with the arm. This constraint can be represented as the following query that derives the fact **panic** whenever the constraint is violated.

$$C: \quad \mathbf{panic} :- \mathtt{arm}(R, Z) \ \& \ \mathtt{obstacle}(O, X, Y) \ \& \ R^2 \geq (X^2 + Y^2) \ \& \ X \geq Z.$$

We assume that the arm is hinged at $[0, 0]$ in the $[X, Y]$ plane. For example, $\mathtt{obstacle}(o, 40, 30)$ says that the point obstacle o is at position $[40, 30]$. $\mathtt{arm}(20, -10)$ says that the arm of robot has radius 20 and that the arm does not go beyond $[-10, Y]$ along the X axis (the line connecting points **a** and **b** in the figure).

In this paper, we consider queries that use arbitrary mathematical functions. We develop theory and algorithms to determine when a tuple subsumes another tuple with respect to a query. We reduce checking for the STS property to a root finding problem and thus show that the STS property depends on the query as well as the subsumed tuple: for a given query only some tuples may satisfy the STS property. We provide a characterization of the STS property for a large class of constraints and also give a sufficient condition for a very general class of constraints.

Paper Outline Section 2 introduces some preliminary notation. Section 3 gives the geometric intuition for single and multiple tuple subsumption. Section 4 describes the class of queries that we will consider. Section 5 has the main results of the paper. We characterize subsumption for the constraint language defined before and give necessary and sufficient conditions for an inserted tuple to satisfy the STS property. We also give sufficient conditions for single tuple subsumption. In the following sections we focus on integrity constraints for simplicity; the results extend to views. We also use the term subsumption henceforth to stand for STS unless otherwise specified.

2 Preliminaries

The integrity constraints of concern in this paper are of the form

$$C: \quad \textbf{panic} :\!-\; l(\bar{Y}) \;\&\; r_1(\bar{Z}_1) \;\&\; r_2(\bar{Z}_2) \;\&\; \ldots \;\&\; r_p(\bar{Z}_p) \;\&\; A.$$

where l is the subgoal with a local predicate (relation), the r_is are subgoals with distinct remote predicates (relations) and A is a conjunction of arithmetic subgoals that relate the attributes of l and r_is No r_i is the same as l or any other r_j. Tuples are inserted into relation `L` and the question of interest is whether some existing tuple(s) in `L` subsumes the inserted tuples such that the truth value of C remains unchanged. If the answer to the above question is "yes", then the remote relations, that represent inaccessible or expensive information, need not be accessed. The query on the local relation that is used to express the question of subsumption is referred to as the local check.

We denote the set $\bar{Z}_1 \cup \ldots \cup \bar{Z}_p$ by $\bar{Z}$. The variables in $\bar{Y}$ and $\bar{Z}$ are distinct. In order to represent shared attributes between l and r_i the corresponding variables in $\bar{Y}$ and $\bar{Z}_i$ are made equal explicitly in A. Thus for each equijoin there is an equality constraint in A.

It is in A that we are most interested. In this paper we consider A, a conjunction of arithmetic constraints, to be of the following form:

$$f_1(\bar{Z})\ op_1\ g_1(\bar{Y}) \ \wedge\ \ldots\ \wedge\ f_k(\bar{Z})\ op_k\ g_k(\bar{Y}).$$

Here the g_is and f_is are functions that take on all real values and are continuous over their domain. The variables $\bar{Y}$ and $\bar{Z}$ are vectors of reals. Each op_i is either $\geq$ or $=$. This choice of op_i is sufficient to model most operators, since we can change the f_is and g_is. Note however, we cannot model the $\neq$ operator.

For clarity of exposition we will assume that op is $\geq$. The analysis is similar when we allow op to be $=$ (Section 5.3). Thus, the arithmetic subgoals A is:

$$f_1(\bar{Z}) \geq g_1(\bar{Y}) \ \wedge\ \ldots\ \wedge\ f_k(\bar{Z}) \geq g_k(\bar{Y}).$$

For a given integrity constraint C, the update we will consider is the insertion of tuple $l(y_1, \ldots, y_n)$ into relation `L`. The inserted tuple is also denote by $l(\bar{y})$.

3 Geometric Intuition for Subsumption

If tuple $l(y_1, \ldots, y_n)$ is inserted into the local relation integrity constraint C may be violated. Tuple $l(\bar{y})$ may be substituted into C to obtain a partially instantiated constraint violation condition $C(\bar{y})$. Consider an instantiation $\bar{z}$ of the variables $\bar{Z}$ that occur in the remote relations. If the instantiation $\bar{z}$ makes $C(\bar{y})$ true, then $\bar{z}$ violates constraint C with tuple $l(\bar{y})$. The set of all such values of variables $\bar{Z}$ that violate the constraint with $l(\bar{y})$ define the "unsafe" region of the remote database given tuple $l(\bar{y})$. The unsafe region defined by a tuple $l(\bar{y})$ is obtained by computing the $g_i(\bar{y})$ in A, resulting in a set of constraints on $\bar{Z}$. Any $\bar{z}$ which satisfies this set of constraints belongs to the unsafe region of $l(\bar{y})$. For example, for the tuple $\mathtt{arm}(20, -10)$ in Example 4, the unsafe region for the remote relation `obstacle` is given by the shaded area in Figure 1. Geometrically, the unsafe region is defined by the surfaces $f_i(\bar{Z}) \geq g_i(\bar{y})$.

For some tuples $l(y_1, \ldots, y_n)$ the unsafe region is empty, which means that no $r(\bar{z})$ can violate integrity constraint C with tuple $l(y_1, \ldots, y_n)$. Such tuples can be inserted into the local database without any constraint checking and are known as *independent updates* [LS93].

If the tuple to be inserted defines a nonempty unsafe region, we must check that there is no remote tuple $r(\bar{z})$ such that $\bar{z}$ lies in the unsafe region. We can query the remote database and determine if such a tuple $r(\bar{z})$ exists. However, we may be able to avoid this remote access if we use the information available in the local database by using subsumption. In particular, if the unsafe regions of a set of existing tuples contain the unsafe region of the inserted tuple, we can conclude that the inserted tuple does not violate constraint C if the constraint was not violated before. The conclusion can be reached as follows. If a remote tuple violates C with the inserted tuple, *i.e.*, lies in the unsafe region for the inserted tuple then the remote tuple also lies in the unsafe region of some existing tuple. Thus, if we assume that the integrity constraint was not violated before the insertion then we are guaranteed that no such remote tuple exists.

The STS property holds if whenever the unsafe region U of the inserted tuple is contained in the unsafe regions of a set of existing tuples then region U is contained completely in the unsafe region of at least one of the existing tuples. This case is of great practical interest, since the subsumption check looks for only one tuple rather than various subsets of an arbitrarily large set of tuples. The check can therefore be represented using first order language constructs.

In general, subsumption depends on the tuple to be inserted: for a given integrity constraint it may be the case that for only some of the inserted tuples subsumption is the complete local check[4]. Subsumption being the complete local check is equivalent to the existence of what we call a "distinguished" point. We show that when the inequalities in the integrity constraint specification are replaced by equalities, and the resulting equations are solved, the existence of a solution implies that subsumption is the complete local check. The solution is a distinguished point. Some of the constraints obtained by the instantiation of A may be "degenerate," *i.e.*, not contribute to the unsafe region. Such degenerate constraints are eliminated from the set of equations, resulting in a different set of equations to be solved. Then subsumption is characterized by there being a solution to the smaller set of equations. We formalize the idea of degenerate

[4] Recall that subsumption refers single tuple subsumption

equations later in the paper.

4 Notation

Given integrity constraint C, and inserted tuple $l(y_1, \ldots, y_n)$ the conjunction of arithmetic subgoals A can be instantiated with $l(\bar{y})$ to obtain the following:

$$f_1(\bar{Z}) \ \geq \ g_1(\bar{y}) \ \wedge \ \ldots \ \wedge \ f_k(\bar{Z}) \ \geq \ g_k(\bar{y}).$$

Let x_i refer to $g_i(\bar{y})$ for $1 \leq i \leq k$. For a given constraint C, we represent the above set of constraints on $\bar{Z}$ by $[x_1, \ldots, x_k]$. If there exists tuples $r_i(\bar{z}_i), 1 \leq i \leq p$ such that $\bar{z}$ satisfies the constraints denoted by $[x_1, \ldots, x_k]$ then inserting tuple $l(\bar{y})$ will violate constraint C.

Definition 1. Region We define $Region(x_1, \ldots, x_k)$ to be the unsafe region associated with $l(\bar{y})$ *i.e.*, the set of all $\bar{z}$ that satisfy the constraints given by $[x_1, \ldots, x_k]$.

$$\bar{z} \in Region(x_1, \ldots, x_k) \Leftrightarrow f_1(\bar{z}) \geq x_1, \ldots, f_k(\bar{z}) \geq x_k.$$

Since we consider only one integrity constraint C, all the unsafe regions in our analysis are obtained by instantiatiating the same A with different values for the variables $\bar{Y}$ *i.e.*, with different tuples of relation `L`. We use the terms $R, S, \ldots$ to refer to these regions. In other words region R is $Region(v_1, \ldots, v_k)$ for some $[v_1, \ldots, v_k]$.

Definition 2. $\geq$ The relationship $\geq$ between different instantiations of A is defined as:

$$[x_1, \ldots, x_k] \geq [v_1, \ldots, v_k] \Leftrightarrow x_1 \geq v_1, \ldots, x_k \geq v_k.$$

Lemma 3.

$$[x_1, \ldots, x_k] \geq [v_1, \ldots, v_k] \Rightarrow Region(x_1, \ldots, x_k) \subseteq Region(v_1, \ldots, v_k).$$

Definition 4. Distinguished point For a region R, $\bar{z} \in R$ is said to be a distinguished point of R if:

$$\forall S, \bar{z} \in S \Rightarrow R \subseteq S.$$

where R and S are regions obtained from the same constraint C.

In Example 4, **a** and **b** are the distinguished points of the unsafe region (refer Figure 1).

Definition 5. Subsumption as the complete local check Consider the inserted tuple $l(\bar{y})$ and let $R = Region(g_1(\bar{y}), \ldots, g_k(\bar{y}))$. Subsumption is the complete local check for this insertion if the following condition holds:

$$R \subseteq S_1 \cup \cdots \cup S_n \Rightarrow \exists j, 1 \leq j \leq n : R \subseteq S_j.$$

where $R, S_1, \ldots, S_n$ are regions obtained from the same constraint C.

5 Results

5.1 Existentially Characterizing Subsumption

Theorem 6. *Consider constraint C of the form defined in Section 2 and let tuple $l(\bar{y})$ be inserted into relation L. Subsumption is the complete local check for C on inserting $l(\bar{y})$ if and only if for $x_i = g_i(\bar{y}), 1 \le i \le k$, $Region(x_1, \ldots, x_k)$ has a distinguished point.*

Theorem 6 is proved in the full version of the paper. It is a very general theorem and does not depend on the expressive power of the constraint language for its validity. The theorem reduces the problem of determining if a k-dimensional unsafe region is contained in a union of m unsafe regions to the problem of determining if a point in k-dimensions is contained in one of m k-dimensional regions. If the problem is indeed reducible, the subsumption (STS) property holds and results in a significant reduction in the complexity of checking subsumption making subsumption an extremely effective optimization in all the applications mentioned in the introduction.

5.2 Computationally Characterizing Subsumption

The following analysis provides a mechanism for identifying the distinguished point for a given class of constraints if such a point exists. In general, the existential characterization of Theorem 6 holds for larger classes of constraints but we may not be able to syntactically identify all these classes. In this section we impose a few restrictions on the constraint language. We require that the g_is be independent of one another. That is, given values for any $g_{i_1}(\bar{Y}), \ldots, g_{i_r}(\bar{Y})$ the possible range of values for $g_i(\bar{Y}), i \notin \{i_1, \ldots, i_r\}$ is unaffected. In other words $g_i(\bar{Y})$ can take on all possible real values, independent of the values of the other g_is. Section 5.3 relaxes the independence assumption.

We now define the Max and Eq functions. Recall that for an inserted tuple $l(\bar{y})$, x_i refers to $g_i(\bar{y}), 1 \le i \le k$.

Definition 7. Max We define the Max transform as follows:

$$Max : [x_1, \ldots, x_k] \rightarrow [p_1, \ldots, p_k] \; .$$

such that

$$Region(p_1, \ldots, p_k) = Region(x_1, \ldots, x_k) \qquad \text{and}$$
$$\forall q_1, \ldots, q_k : \{Region(q_1, \ldots, q_k) = Region(x_1, \ldots, x_k) \Rightarrow .$$
$$[p_1, \ldots, p_k] \ge [q_1, \ldots, q_k]\}.$$

Geometrically we can think of $Region(x_1, \ldots, x_k)$ as being the "unsafe" region defined by the surfaces $f_i(\bar{Z}) \ge x_i$. Now some of these surfaces (constraints) may be degenerate in that they may not restrict the "unsafe" region at all. The Max transform moves these degenerate surfaces to the maximum extent possible without affecting the original unsafe region $Region(x_1, \ldots, x_k)$. Intuitively, it corresponds to having these surfaces just "touch" (be tangential to) the unsafe region. For instance, consider the robot arm of Example 4. If tuple `arm`$(1, -3)$ is inserted then the unsafe region $[1, -3]$ is the shaded circle in Figure 2(A).

However, the line $x = -3$ does not restrict the region. If the Max transform is applied to $[1, -3]$, the line moves to $x = -1$ as shown by the dotted line in Figure 2(B). $Max([1, -3])$ is computed to be $[1, -1]$.

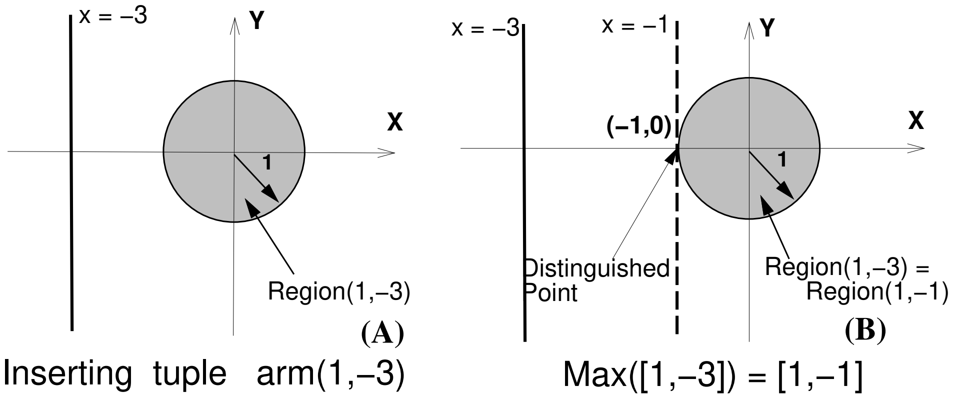

Fig. 2.

Definition 8. Eq We define $Eq([x_1, \ldots, x_k])$ to be the set of equations:

$f_i(\bar{Z}) = x_i$.

We say that $Eq([x_1, \ldots, x_k])$ has a solution if there is some $\bar{Z} = \bar{z}$ such that $Eq([x_1, \ldots, x_k])$ is true.

Theorem 9. *Consider constraint C of the form defined in Section 2 with the restriction that* `op` *is $\geq$ and the g_is are independent. Let tuple $l(\bar{y})$ be inserted into relation L. Subsumption is the complete local check iff $Eq(Max([x_1, \ldots, x_k]))$ has a solution $\bar{z}$, and for $x_i = g_i(\bar{y}), 1 \leq i \leq k$, $\bar{z} \in Region(x_1, \ldots, x_k)$.*

Proof. The intuition for both directions of the proof is based on $\bar{z}$ being a distinguished point of $Region(x_1, \ldots, x_k)$. The details are given in the full version of the paper.

5.3 Extensions

We now relax the restrictions imposed on the set of arithmetic inequalities in constraint C and discuss how Theorem 9 is affected. First we consider the case when the functions g_is in the set of arithmetic inequalities A are not independent. In Section 5.3 we consider the case when the g_is are bounded and may not take all values on the real number line. In Section 5.3 we consider the case when the constraint uses equalities in its set of arithmetic inequalities, *i.e.*, the constraint uses joins. Finally, in Section 5.3 we state a sufficient condition for subsumption. Even though we may not have necessary and sufficient conditions, sufficient conditions are often very useful in using subsumption for constraint checking, view maintenance, or query optimization.

Dependent g_is Theorem 9 is dependent on the constraint language and is contingent on the independence of the g_is, *i.e.*, if the g_is are not independent the theorem does not hold. The existential characterization that ties STS with the existence of a distinguished point still holds. In addition, it is still the case that for inserted tuple $l(\bar{y})$ if $Eq(Max([x_1, \ldots, x_k])$ has a solution $\bar{z} \in Region(x_1, \ldots, x_k)$, subsumption is the complete check. The proof of this statement is identical to that for Theorem 9.

However, the converse, that $Eq(Max([x_1, \ldots, x_k])$ *has to have* a solution if subsumption is the complete check, relies on the independence of the g_is. When the g_is are not independent it can be the case that subsumption is the complete local check and yet there is no solution.

Example 5. An example is the following integrity constraint:

C: **panic** :- $l(X)$ & $r(Z)$ & $Z \geq -X$ & $Z \leq X$.

If we wish to insert the local tuple $l(x)$, the relevant equations are

$Z = -x$ and $-Z = -x$.

These equations have a solution only when $x = 0$ and the solution is $Z = 0$ *in* $Region(0, 0)$. Theorem 9 would indicate that subsumption is the complete local check only when inserting $l(0)$. Actually, subsumption is the complete local check for the insertion of any tuple into the local database.

Bounded g_is In some cases we may want to bound the values of the functions g_is to be in a fixed interval $[a_i, b_i]$ rather than let the values range from $[-\infty, \infty]$. In Example 1 for instance, we may wish to add the constraint that all salaries are positive.

We can augment the original set of arithmetic subgoals A with the subgoals $-a_i \geq -g_i$ and $b_i \geq g_i$ for each g_i, to get a new set of subgoals A'. A' satisfies the format of arithmetic subgoals we have used thus far, since the lower and upper bound subgoals have left hand sides that are merely constant functions of the remote variables $\bar{Z}$. Let us order the subgoals of A' as follows: the first k subgoals are identical to those in A. The $k+1$ and $k+2$ subgoals correspond to the lower and upper bounds of g_1. The $k+3$ and $k+4$ subgoals correspond to the lower and upper bounds of g_2 and so on. In general the lower and upper bound subgoals for g_i are subgoals $k+2i-1$ and $k+2i$ respectively. If A has k subgoals, A' has $3k$ subgoals. When we wish to insert a new tuple $l(\bar{y})$ we instantiate A' with the $g_i(\bar{Y})$ values. Now the set of arithmetic constraints is given by $[x_1, \ldots, x_k, -x_1, x_1, \ldots, -x_k, x_k]$, where $x_i = g_i(\bar{y})$.
From the discussion given above about dependent g_is (which is the case here), if

$$Eq(Max[x_1, \ldots, x_k, -x_1, x_1, \ldots, -x_k, x_k])$$

has a solution in the corresponding region, subsumption holds. It can be shown that

$$Eq(Max[x_1, \ldots, x_k, -x_1, x1, \ldots, -x_k, x_k]) = Eq(Max[x_1, \ldots, x_k])$$

since the last $2k$ subgoals have constants on their right hand sides. Hence it appears that bounded g_is behave the same as unbounded g_is. However, that is not the case.

Example 6. Consider the following query:

$$C: \quad \mathbf{panic} :\!-\ l(X,Y)\ \&\ r(Z)\ \&\ Z \geq X\ \&\ -Z \geq -Y.$$

If we wish to insert a tuple $l(x,y)$ with $x < y$, we check if $Eq(Max[x,-y])$ has a solution (note, in this case $Max[x,-y] = [x,-y]$, since neither subgoal is degenerate). Clearly there is no solution and hence we conclude subsumption does not hold. This statement is true if X and $-Y$ are unbounded. Consider now X to be restricted to the interval $[a,b]$. If we now wish to insert $l(b,y)$ with $b < y$ we have subsumption, even though $Eq(Max[b,-y])$ has no solution.

Thus when the g_is are bounded, Theorem 9 does not hold as specified. The definitions of Max and Eq need to be modified for the characterization of Theorem 9 to hold.

Let the upper bound for each g_i be b_i (the lower bound does not matter).

$Max : [x_1,\ldots,x_k] \rightarrow [p_1,\ldots,p_k]$ is defined as before except if a $p_i \geq b_i$, we replace p_i by the corresponding b_i. This guarantees that $Max([x_1,\ldots,x_k]) \leq [b_1,\ldots,b_k]$ and thus is legal. $Eq([x_1,\ldots,x_k])$ is now defined to be the set of equations:

$$f_i(\bar{Z}) = x_i \quad if\ x_i \leq b_i.$$
$$TRUE \quad if\ x_i = b_i.$$

In some sense a subgoal of A does not contribute to the subsumption check if the inserted tuple has the maximum possible value for the corresponding g_i.

With these enhanced definitions of Max and Eq, Theorem 9 holds. The proof is similar and is not stated here.

Joins Until now we considered arithmetic inequalities of the form $f_i(\bar{Z})\ \texttt{op}\ g_1(\bar{y})$ where `op` was restricted to be $\geq$. Thus joins were not expressible because implicit equalities were not allowed in the constraint specification. In this section we show that the previous restriction was needed only for the sake of simplicity and that the results extend naturally to the case of joins.

In Example 1 we require that an employee's department be the same as that of a manager to compare their salaries. Since we do not allow shared attributes, we rewrite the constraint using an explicit "=" to represent the join.

$$C: \quad \mathbf{panic} :\!-\ \texttt{emp}(E, D_E, S)\ \&\ \texttt{dept}(D_M, MS)\ \&\ D_E = D_M\ \&\ S \geq MS.$$

When we permit *op* to be either $\geq$ or $=$, without loss of generality we can write the arithmetic subgoals A to be:

$$f_1(\bar{Z}) = g_1(\bar{Y})\ \wedge\ \ldots\ \wedge\ f_i(\bar{Z}) = g_i(\bar{Y})\ \wedge$$
$$f_{i+1}(\bar{Z}) \geq g_{i+1}(\bar{Y})\ \wedge\ \ldots\ \wedge\ f_k(\bar{Z}) \geq g_k(\bar{Y}).$$

i.e. the first i subgoals are the equality constraints and the remaining subgoals are inequalities. Note, now:

$$\bar{z} \in Region(x_1,\ldots,x_k) \Leftrightarrow f_1(\bar{z}) = x_1,\ldots,f_i(\bar{z}) = x_i, f_{i+1}(\bar{z}) \geq x_{i+1},\ldots,f_k(\bar{z}) \geq x_k$$

and

$$[x_1, \ldots, x_k] \geq [v_1, \ldots, v_k] \Leftrightarrow x_1 = v_1, \ldots, x_i = v_i, x_{i+1} \geq v_{i+1}, \ldots, x_k \geq v_k.$$

Theorem 9 holds for the above class of constraints also, and the proof is the same as before (see full version of paper).

A Sufficient Condition For Subsumption We give below a sufficient condition for subsumption to be the complete check. The only requirement for it to hold is that the arithmetic subgoal, A, can be written in the form given in Section 2. There are no restrictions on the g_is for the above lemma to hold.

Lemma 10. *Consider the integrity constraint C:*

$$C: \quad \textbf{panic} :- l(\bar{Y}) \ \& \ r_1(\bar{Z}_1) \ \& \ r_2(\bar{Z}_2) \ \& \ \ldots \ \& \ r_p(\bar{Z}_p) \ \& \ A.$$

where A is

$$f_1(\bar{Z} \ op_1 \ g_1(\bar{Y}) \ \wedge \ \ldots \ \wedge \ f_k(\bar{Z}) \ op_k \ g_k(\bar{Y}).$$

Let $l(\bar{y})$ be a tuple inserted into relation `L` *and let $x_i = g_i(\bar{y}), 1 \leq i \leq k$. Subsumption is the complete local check for the inserted tuple if the set of equations:*

$$f_1(\bar{Z} \ = \ g_1(\bar{Y}) \ \wedge \ \ldots \ \wedge \ f_k(\bar{Z}) \ = \ g_k(\bar{Y}).$$

has a solution $\bar{z}$ in $Region(x_1, \ldots, x_k)$,

Proof. Let $Region(x_1, \ldots, x_k) \subseteq Region(v_1^1, \ldots, v_k^1) \cup \ldots \cup Region(v_1^n, \ldots, v_k^n)$ where each $\bar{v}^j$ represents a tuple of relation `L`. That is, the multiple subsumption property holds for the inserted tuple $l(\bar{y})$. Now consider a particular point $\bar{z}$ in the unsafe region for $l(\bar{y})$.

$\bar{z} \in Region(x_1, \ldots, x_k) \Rightarrow \exists j : \bar{z} \in Region(v_1^j, \ldots, v_k^j)$.
% i.e., $\bar{z}$ must be in the unsafe region for some tuple $\bar{v}^j$.
a: $\Rightarrow \forall i, f_i(\bar{z}) \geq v_i^j$ *% that is, the arithmetic subgoals A must be satisfiable given $\bar{z}$ and $\bar{v}^j$.*

If the conditions of the lemma hold, then $f_i(\bar{z}) = x_i$. Therefore, substituting x_i for $f_i(\bar{z})$ in (a) we obtain:

$\forall i, x_i \geq v_i^j$
$\Rightarrow [x_1, \ldots, x_k] \geq [v_1^j, \ldots, v_k^j]$
$\Rightarrow Region(x_1, \ldots, x_k) \subseteq Region(v_1^j, \ldots, v_k^j)$.

Thus subsumption is the complete local check.

6 An Example

Consider the integrity constraint $C1$:

$$\begin{aligned} C1: \quad \textbf{panic} :- \ & \texttt{emp}(D, S_E, OT, Rate) \ \& \ \texttt{manager}(D, S_M, Bonus) \ \& \\ & S_M \geq S_E \ \& \ Bonus \leq OT * Rate. \end{aligned}$$

We can rewrite $C1$ this in the our constraint language as follows:

$$\begin{aligned} C1: \quad \textbf{panic} :- \ & \texttt{emp}(D_E, S_E, OT, Rate) \ \& \ \texttt{manager}(D_M, S_M, Bonus) \ \& \\ & D_M = D_E \ \& \ S_M \geq S_E \ \& \ -Bonus \geq -OT * Rate. \end{aligned}$$

Now let's say we want to insert emp("*sales*", $30000, 400, 30)$ into the local emp database. We first compute the g_is (the functions that appear on the right-hand-side of the arithmetic subgoals above):

g_1(emp("*sales*", $30000, 400, 30$)) = "*sales*".
g_2(emp("*sales*", $30000, 400, 30$)) = 30000.
g_3(emp("*sales*", $30000, 400, 30$)) = -12000.

Now we check if there can be a manager tuple that can cause a constraint violation; we check to see if the unsafe region *Region*("*sales*", $30000, -12000$) is empty. In this case we find it is not empty: A sales manager who earns more than 30000 but who has a bonus less than 12000 would cause a violation. If the unsafe region was empty we could have gone ahead and inserted the tuple without any further checks. Thus, we proceed by checking if subsumption is the complete local check. We find it is, since the solution D_M = "*sales*", $S_M = 30000$ and $Bonus = 12000$ satisfies Lemma 10. We now query the emp database to find if there is any emp($D_E, S_E, OT, Rate$), such that $[D_E, S_E, -OT * Rate] \leq$ ["*sales*", $30000, -12000$]. If there is one such tuple (a sales employee who earned less than 30000 but made more than 12000 in overtime wages) already in the database, then inserting the new tuple will not violate constraint $C1$. If no such tuple exists in the DB, then we require a conventional constraint check that queries the manager relation.

For this constraint $C1$, we can determine that subsumption is the complete check for all tuples to be inserted. So we can parameterize the inserted tuple and derive an SQL query at compile time such that this query is the complete local check for subsumption.

7 Conclusions

We considered the property of subsumption, *i.e.*, when can a tuple be ignored with respect to a query given some other tuples in the same relation. Subsumption finds many uses: in intelligent caching, in query optimization, in optimizing integrity constraint checking, and in efficient incremental view maintenance. We illustrate these uses and then give the intuition behind subsumption. We consider multiple tuple subsumption (when many tuples together subsume one tuple) and single tuple subsumption (when just one tuple is sufficient to subsume another tuple). We are especially interested in the latter case because it is the more practical case and can be implemented using first order languages like SQL without recursion.

We consider integrity constraints, 0-ary queries, in this paper in order to simplify our analysis. For a large class of such queries we characterize subsumption in terms of the existence of a "distinguished point". This characterization is a powerful result as it reduces the problem of determining if a k-dimensional unsafe region is contained in a union of m unsafe regions to the problem of determining if a *point* in k-dimensions is contained in one of m k-dimensional regions.

The above characterization enables us to identify classes of queries that use arbitrary mathematical functions as subgoals and for which subsumption is computationally feasible. That is, the existential characterization holds for a larger class of constraints than those for which we can compute the distinguished point. This result enables us to generate automatic query rewrite and view maintenance algorithms for such queries.

By providing a formal basis for subsumption we hope to identify, apply and possibly unify optimization techniques in the many areas mentioned in the introduction. In particular, query optimization is a promising application for subsumption, since we can now automatically generate aggregate subgoals during query rewriting. We are also currently exploring how results from computational geometry can yield efficient subsumption checking algorithms. The intuition in Section 3 gives an idea of how geometry results tie into subsumption. We are looking at other ways of correlating subsumption with properties of mathematical functions. For instance, the linearity of functions, number of variables, independence of the variables, etc.

Acknowledgements

We would like to thank Anand Rajaraman, Jeff Ullman, and Jennifer Widom for valuable comments and feedback on the ideas presented in this paper. The authors' research was supported by NSF grants IRI–91–16646 and IRI–92–23405, by ARO grant DAAL03–91–G–0177, and by Air Force Grant F33615–93–1–1339.

References

[CS94] Surajit Chaudhuri and Kyuseok Shim. Including Group-By in Query Optimization. To appear in *Proceedings of the Twentieth International Conference on Very Large Databases (VLDB)*, 1994.

[GSUW94] Ashish Gupta, Shuky Sagiv, Jeffrey D. Ullman, and Jennifer Widom. Constraint Checking with Partial Information. In *Proceedings of the Thirteenth Symposium on Principles of Database Systems (PODS)*, 1994.

[GW93] Ashish Gupta and Jennifer Widom. Local Checking of Global Integrity Constraints . In *Proceedings of ACM SIGMOD 1993 International Conference on Management of Data*, pages 49–59.

[LS93] A.Y. Levy and Y. Sagiv. Queries independent of updates. In *Proceedings of the Nineteenth International Conference on Very Large Data Bases*, pages 171–181, Dublin, Ireland, August 1993.

[TH93] Sanjai Tiwari and H. C. Howard. Constraint Management on Distributed AEC Databases. In *Fifth International Conference on Computing in Civil and Building Engineering*, pages 1147–1154. ASCE, 1993.

[Ull89] J. D. Ullman. *Principles of Database and Knowledge-Base Systems*, volume 2. Computer Science Press, New York, 1989.