Magic-sets Transformation in Nonrecursive Systems

Extended Abstract

Ashish Gupta*
Computer Science Department
Stanford University
Stanford, CA 94305, USA.
agupta@cs.stanford.edu

Inderpal Singh Mumick
AT&T Bell Laboratories
600 Mountain Avenue
Murray Hill, NJ 07974, USA.
mumick@research.att.com

Abstract

A nonrecursive system is any database system whose query language does not support recursive queries. Thus, many existing commercial SQL database systems are nonrecursive systems. Query optimization is an important issue for nonrecursive queries, and the magic-sets transformation has been shown to improve the performance of nonrecursive queries by many orders of magnitude [MFPR90]. It is thus important to use the magic-sets transformation in nonrecursive systems.

However, there is a problem. The magic-sets optimization can transform a nonrecursive query into a recursive query. Since a recursive query cannot be executed by a nonrecursive system, such a transformation is fatal. The magic-sets transformation cannot therefore be used in a nonrecursive system.

In this paper we present algorithms that achieve the optimization of the magic-sets transformation while guaranteeing that the transformed program will be nonrecursive whenever the original program is nonrecursive. The algorithms can be extended to the supplementary magic-sets transformation. We also define a new optimization technique for recursive and nonrecursive queries, *covered subgoal elimination*, that can eliminate subgoals from a rule, and can sometimes convert a recursive query into a nonrecursive one.

The algorithms presented in this paper are of practical relevance since they make it possible to incorporate the magic-sets transformation into existing commercial database systems.

*Work was supported by an NSF grant IRI-90-16358, an Air Force grant AFOSR-88-0266, and a grant of IBM Corporation.

1 Introduction

The Problem: Most existing database systems (IBM's DB2, for example) do not support recursive queries. SQL is the universal query language amongst relational databases, and the currently implemented SQL standard (SQL2, [ISO90]) does not provide for recursion. We will call a database system nonrecursive if its query language does not permit recursive queries. Thus, DB2 is a nonrecursive system.

The magic-sets transformation ([BR87, Ull89]) is a general purpose query optimization technique initially proposed for recursive queries in deductive databases. It has been shown that the magic-sets transformation (MST) can be extended to optimize relational SQL queries where SQL has been extended to permit recursion [MFPR90, MPR90, Mum91]. Performance experiments have demonstrated that MST is an invaluable optimization for nonrecursive queries [MFPR90, Mum91]. It is thus imperative that MST be used to optimize queries in a nonrecursive system.

However, there is one problem that must be solved before MST can be used in a nonrecursive system. MST has the undesirable property that it can transform a nonrecursive query into a recursive query, as in Example 1.1 below.

EXAMPLE 1.1 Consider query Q on the nonrecursive program P (Note that we use a letter, such as P, to represent a program comprised of rules $P1, P2, \ldots$):

(Q): ?- $t(0)$.

$(P1)$: $t(X)$:- $s(X,Y)$ & $p(Y)$.

$(P2)$: $s(X,Y)$:- $p(X)$ & $q(X,Y)$.

$(P3)$: $p(X)$:- $e_1(X,Y)$ & $e_2(Y)$.

$(P4)$: $q(X,Y)$:- $e_3(X,Z)$ & $e_4(Z,Y)$.

Applying the magic-sets transformation to program P, we derive the program MP below:

$(MP1)$: $t(X)$:- $m\_t(X)$ & $s(X,Y)$ & $p(Y)$.

$(MP2)$: $s(X,Y)$:- $m\_s(X)$ & $p(X)$ & $q(X,Y)$.

$(MP3)$: $p(X)$:- $m\_p(X)$ & $e_1(X,Y)$ & $e_2(Y)$.

$(MP4)$: $q(X,Y)$:- $m\_q(X)$ & $e_3(X,Y)$ & $e_4(Z,Y)$.

$(MP5)$: $m\_t(0)$.

$(MP6)$: $m\_s(X)$:- $m\_t(X)$.

$(MP7)$: $m\_p(Y)$:- $m\_t(X)$ & $s(X,Y)$.

$(MP8)$: $m\_p(X)$:- $m\_s(X)$.

$(MP9)$: $m\_q(X)$:- $m\_s(X)$ & $p(X)$.

Program MP is recursive since predicate m_p depends upon predicate s (rule $MP7$), s depends upon predicate p (rule $MP2$), and p depends upon predicate m_p (rule $MP3$). The recursion arises because rule $P1$ asks for information to be passed sideways from predicate s into predicate p, while rule $P2$ defines predicate s using predicate p. $\Box$

A transformation that produces recursive queries cannot be used in a system that does not support recursive queries. Can we somehow avoid introducing recursion when doing the magic-sets transformation? If recursion could be avoided, the magic-sets transformation would be usable in nonrecursive systems, and thus be a very attractive optimization for all existing relational database systems.

The Solution: We first show that recursion introduced by the magic-sets transformation is bounded [Nau86], and present a simple algorithm for avoiding recursion by replicating all common subexpressions in the original program. The algorithm is called RACS/MST (*Replicate All Common Subexpressions and apply Magic-Sets Transformation*). However, each replication of a common subexpression leads to replicated work, and it would be preferable not to duplicate all common subexpressions.

We develop two algorithms for nonrecursive magic-sets transformation that are more efficient than RACS/MST. We also define a new optimization method, *covered subgoal elimination* (CSE), that simplifies rules by eliminating certain subgoals it infers to be redundant. Covered subgoal elimination applies to both recursive and nonrecursive queries, and can sometimes convert a recursive query into a nonrecursive query.

The first nonrecursive magic-sets transformation, MST/CSE (*Magic-Sets Transformation followed by Covered Subgoal Elimination*), eliminates recursion introduced by the magic-sets transformation through selective unfolding [TS84] followed by covered subgoal elimination. The second nonrecursive magic-sets transformation, RLCS/MST (*Replicate Limited Common Subexpressions and apply Magic-Sets Transformation*) does limited replication of common subexpressions before applying the magic-sets transformation.

Relevance: Existing commercial database systems do not support recursive queries. Recursive queries are not needed in many traditional applications, and nonrecursive database systems will continue to be developed. Our work makes it possible to incorporate the magic-sets transformation into nonrecursive systems.

As proposed in (SQL3, [ISO90]), SQL is being extended to permit recursion and hopefully most future database systems will support recursion. However, even in recursive systems it is desirable not to transform a nonrecursive query into a recursive one since it is expected that recursive queries will be less efficient to execute than equivalent nonrecursive queries.

Covered subgoal elimination is an important optimization method in its own right, even though its use in this paper is limited to developing a nonrecursive magic-sets transformation.

Paper Layout: The paper is organized as follows. After some preliminary definitions in Section 1.1, we define a new graphical structure, the *information passing graph* (IPG) in Section 2. An IPG represents the sideways passing of information (join predicates) between subgoals in a rule together with the dependencies from the subgoals to the head of a rule, and is a very useful tool to reason about recursion introduced by MST. Section 3 presents the RACS/MST algorithm based on replication of *all* common subexpressions, and shows that the recursion introduced by MST is bounded. Covered subgoal elimination is presented in Section 4, and is used to develop the MST/CSE algorithm for nonrecursive magic-sets transformation in Section 5. The RLCS/MST algorithm based on *limited* replication of common subexpressions is discussed in Section 6. We end with conclusions and a comparison between the two nonrecursive magic-sets transformations in Section 7.

1.1 Background

In this paper we will write database queries in datalog [Ull89]. Datalog is the language of horn clause logic rules, written as "if-then" rules. A datalog rule is required to be range-restricted: it must define a *finite* relation for the predicate in the head of the rule. In this paper we will concentrate on nonrecursive queries. Mumick [Mum91] shows that nonrecursive datalog programs can be expressed in SQL using views, and that SQL queries can be expressed in datalog extended with duplicates and grouping with aggregation. Thus, the algorithms developed in this paper can be mapped to relational database systems having SQL as their query language.

We will use dependency graphs to represent the dependence between predicates in a datalog program.

Definition 1.1 Dependency Graph (DG): The dependency graph, $\mathsf{DG}(P)$, for a datalog program P is a directed graph built as follows: for every ordinary predicate p in program P, there is a node labeled p. There is an edge from node q to node p (q and p may be the same node) if there is a rule of the form:

(r): p :- ... & q & ....

□

EXAMPLE 1.2 Figure 1 shows the dependency graph for program P introduced in Example 1.1. □

Figure 1: Dependency Graph

A datalog program is recursive if and only if its dependency graph has a cycle, for then the predicates in the cycle are defined in terms of each other. In this paper we concentrate on nonrecursive programs, and assign stratum numbers to predicates as follows:

Definition 1.2 Stratum Numbers: In a nonrecursive program P, the stratum number of a base predicate is 0, and the stratum number of a derived predicate is one more than the largest stratum number of all its children in the dependency graph. □

Our definition of strata differs from the definition given by Ullman ([Ull89], Chapter 3.6). Given two mutually nonrecursive predicates p and q in a datalog program (without negation), Ullman would place both predicates into the same stratum. However, we place predicates p and q in different strata. If predicate p depends on predicate q, then predicate p is placed in a higher stratum. If predicates p and q do not depend on each other, then it is possible for the predicates to be placed in the same stratum.

We assume the reader is familiar with the concepts of sips, adornments, magic-sets, and conjunctive query containment covered in [Ull89]. A sideways information passing strategy (sips) is a decision on how to pass information sideways in the body of a rule while evaluating the rule. The magic-sets transformation is defined on adorned programs, and is guided by a sips. The adornments are used to identify the bound arguments of each subgoal in the rule. To simplify the treatment in this paper, we will work on unadorned programs, and we will assume we know the bound arguments of each predicate.

The following definition of the magic-sets transformation is as described in [BR87], simplified only to omit the adornments.

Definition 1.3 The Magic-sets Transformation (MST): Given a program P, a query goal q, and a full sips for each rule for program P, the magic-sets transformation derives a magic program MP as follows:

1. Create a new predicate m_p (the magic predicate for p) for each derived predicate p in program P. The arity of m_p is the number of bound arguments of p.

2. For each rule r in program P, add a *modified version* of rule r to program MP. If rule r has head $p(\bar{t}\,)$, where $\bar{t}$ is shorthand for all arguments of p, the modified version is obtained by adding the literal $m\_p(\overline{t^b})$ into the body of rule r as the first subgoal, where $\overline{t^b}$ denotes all bound arguments of $p(\bar{t}\,)$.

3. For each rule r in program P with head $p(\bar{t}\,)$, and for each subgoal $q_i(\overline{t_i}\,)$ in its body such that q_i is a derived predicate, add a *magic rule* to program MP. The head is $m\_q_i(\overline{t_i^b})$. The body contains all subgoals that precede q_i in the sips order associated with r, as well as the literal $m\_p(\overline{t^b})$.

4. Create a *seed* fact $m\_q(\bar{c})$, where $\bar{c}$ is the set of constants equated to the bound arguments of the query goal.

□

Figure 2: Information Passing Graph for program P of Example 1.1

2 Information Passing Graphs

Dependency graphs capture the "upwards" flow of information from subgoals to the predicates they derive. Information can also be passed sideways from one subgoal into another in the same rule, as specified by a sips. The magic-sets transformation introduces magic predicates to pass information sideways. We define information passing graphs to represent this sideways flow of information for a given sips, in addition to the "upward" flow already represented by a dependency graph. We use the following definition of a subgoal position.

Definition 2.1 Subgoal Position (SP)**:** Consider a rule r of the form:

$$(r): \quad q_0 :- q_1 \ \& \ q_2 \ \& \ \ldots \ \& \ q_n.$$

where built-in predicates have been ignored, and $q_1, \ldots, q_n$ are ordinary predicates. The subgoal position of the predicate q_i in rule r, written $\mathsf{SP}(q_i, r)$, is i. □

Definition 2.2
Information Passing Graphs (IPG)**:** The information passing graph of a datalog program P, $\mathsf{IPG}(P)$, is a directed labeled graph obtained by extending the dependency graph. The IPG has a node for every predicate q in program P, and an additional magic node mq for every *derived* predicate q in program P. When the derived predicate q occurs in a rule r, define $\mathsf{SP}(mq, r) = \mathsf{SP}(q, r)$. The IPG has two types of edges: dependency (**D**) edges denoted by $\rightarrow$ and information (**I**) edges denoted by $\Rightarrow$. Given rules of form r (as in Definition 2.1), the edges of the information passing graph are derived as follows:

1. A dependency edge $q_i \rightarrow q_0$ is derived from every rule of form r.

2. An information edge $mq_0 \Rightarrow mq_j$, is derived from every rule of form r, where q_j is the first derived subgoal in rule r

3. Given a subgoal q_i in a rule of form r, let q_j be the first derived subgoal (if one exists) after subgoal q_i. The information edge $q_i \Rightarrow mq_j$, is derived from every subgoal q_i in every rule r, provided a derived subgoal follows q_i.

4. An information edge $mq_0 \Rightarrow q_0$ is derived from every rule of form r.

A particular **I** or **D** edge can be derived more than once, from different rules, or from different occurrences of a predicate in the body of a rule. An edge is labeled to indicate all its derivations. Formally, an edge label is a set of tuples. If a rule r derives an edge from node u to node v due to an occurrence of predicate u at position $\mathsf{SP}(u, r)$ and predicate v at position $\mathsf{SP}(v, r)$, the edge label contains the tuple $< r, \mathsf{SP}(u, r), \mathsf{SP}(v, r) >$. □

EXAMPLE 2.1 Figure 2 shows the information passing graph of program P introduced in Example 1.1. Magic nodes mq, mp, ms and mt correspond to the derived predicates q, p, s, and t respectively. In rule $P1$, $p(Y)$ is the first derived predicate after subgoal $s(X, Y)$, deriving the information edge $s \Rightarrow mp$. Since $\mathsf{SP}(s(X,Y), P1) = 1$ and $\mathsf{SP}(p(Y), P1) = 2$, the edge $s \Rightarrow mp$ has the label $< P1, 1, 2 >$. The edge $ms \Rightarrow mp$ is derived from rule $P2$, and has the label $< P2, 0, 1 >$. In this paper we will ignore labels on outedges from magic nodes, since these labels do not have a bearing on our discussions. □

I edges in an information passing graph represent the sideways flow of bindings between subgoals in a rule, and the downward flow of bindings from the head of a rule into the body of a rule. The magic-sets transformation uses this information passing to derive the magic program MP. We can relate a magic node mp in the information passing graph of a program P to the magic predicate m_p in program MP. Paths in $\mathsf{IPG}(P)$ then represent paths in the dependency graph of program MP.

Theorem 2.1 *Let MP be the program obtained by magic-sets transformation of a given nonrecursive program P. Let* $\mathsf{DG}(MP)$ *be the dependency graph of program MP, and let* $\mathsf{IPG}(P)$ *be the information passing graph of program P. Relate the node for a magic predicate m_p in* $\mathsf{DG}(MP)$ *to the magic node mp in* $\mathsf{IPG}(P)$*. Then, there is a path between two nodes in* $\mathsf{IPG}(P)$ *if and only if there is a path between the corresponding nodes in* $\mathsf{DG}(MP)$*.* □

From Theorem 2.1 it follows that the the magic-sets transformation of a nonrecursive program is recursive if and only if the information passing graph of the program has a cycle.

Definition 2.3 Simple Cycle: A simple cycle is a cycle in which no node is repeated. □

A node in an information passing graph that is not a magic node will be referred to as a non-magic node. Every non-magic node has a stratum number obtained from the dependency graph (Definition 1.2). Magic nodes in an information passing graph do not have a stratum number.

Lemma 2.1 *Every cycle in the information passing graph of a nonrecursive program must have at least one non-magic node.* □

Definition 2.4 Cycle Stratum Number (CSN): The cycle stratum number of a cycle C, $\mathsf{CSN}(C)$, is the minimum stratum number amongst all the non-magic nodes in cycle C. □

Cycle Stratum numbers for cycles in information passing graphs are well defined since, by Lemma 2.1, every cycle in an information passing graph has at least one non-magic node.

Cycle Identifier	Elements in Order of Traversal	CSN	Nodes in Lowest Stratum
$C1$	p, mq, q, s, mp, p	1	p, q
$C2$	p, s, mp, p	1	p

Table 1: Simple cycles.

EXAMPLE 2.2 Table 1 lists the cycles in the information passing graph shown in Figure 2. Cycle $C1$ has three non-magic nodes: p, q, and s. Nodes p and q are in stratum 1, and node s is in stratum 2. The cycle stratum number for cycle $C1$ is thus 1. □

The following properties of cycles in information passing graphs are useful.

Lemma 2.2 *Let C be a cycle in the information passing graph of a nonrecursive program P, and let q be a non-magic node in cycle C with the minimal stratum number* ($\mathsf{CSN}(C)$)*. Let MP be the magic-sets transformation of program P. Then:*

- *The edge of cycle C leading into node q is an information edge.*
- *Predicate q is a derived predicate.*
- *The magic program MP includes the magic predicate m_q.*

□

3 RACS/MST

This section discusses a simple method to modify a program so that its magic-sets transformation is guaranteed to be nonrecursive. The method relies on common subexpression elimination. If a derived predicate is used in two subgoals, the derived predicate is said to form a common subexpression. Nonrecursive programs can be classified into two types depending on the presence or absence of common subexpressions.

Definition 3.1 Hierarchical Program: A nonrecursive program with no common subexpressions is said to be hierarchical. □

Definition 3.2 Directed Acyclic Graph Program: A nonrecursive program with common subexpressions is said to be a directed acyclic graph (dag) program. □

Figure 3: Information Passing Graph of program P_h

Theorem 3.1 *Let P be a hierarchical program, and let MP be the program derived by a magic-sets transformation of program P. Then program MP is nonrecursive.* □

Theorem 3.1 suggests the following algorithm for applying the magic-sets transformation to a dag program so as to derive a nonrecursive magic program.

Algorithm 3.1 RACS/MST: *Given a dag program P, transform program P as follows:*

1. *Remove all common subexpressions in program P by replicating any predicate used in two or more subgoals. Let P_h be the resulting hierarchical program.*

2. *Apply magic-sets transformation to program P_h. Let MP be the resulting magic program.*

◇

Algorithm 3.1 is named RACS/MST, short for *Replicate All Common Subexpressions and apply Magic-Sets Transformation.* Theorem 3.1 guarantees that the output program MP of Algorithm 3.1 is nonrecursive.

Corollary 3.2 *Algorithm 3.1 derives a nonrecursive program.* □

EXAMPLE 3.1 We will apply Algorithm 3.1 to the program P introduced in Example 1.1. Predicate p is the only common subexpression, and it is replicated into two predicates: p and p_1. The resulting program P_h is:

(Q_h): ?- :– t(0).

(P_h1): $t(X)$:– $s(X,Y)$ & $p(X)$.

(P_h2): $s(X,Y)$:– $p1(X)$ & $q(X,Y)$.

(P_h3): $p(X)$:– $e_1(X,Z)$ & $e_2(Z)$.

(P_h4): $p1(X)$:– $e_1(X,Z)$ & $e_2(Z)$.

(P_h5): $q(X,Y)$:– $e_3(X,Z)$ & $e_4(Z,Y)$.

Figure 3 shows the information passing graph of program P_h. Since $\mathsf{IPG}(P_h)$ is acyclic, the magic-sets transformation of program P_h is nonrecursive (Theorem 2.1).

□

Algorithm 3.1 implies the following theorem.

Theorem 3.3 *Let P be a dag program, and let MP be the program derived by a magic-sets transformation of program P. Then if program MP is recursive, the recursion is bounded [Nau86].* □

Since every nonrecursive program is either hierarchical or dag, RACS/MST can be used to apply the magic-sets transformation in a nonrecursive system. However, it is not desirable to replicate all common subexpressions in a program, since each replication causes computation to be repeated. Can we do better than RACS/MST? Can we avoid replicating all common subexpressions?

4 Covered Subgoal Elimination

EXAMPLE 4.1 (**Motivation**): Consider the program S:

$(S1)$: $s(X,Y)$:- $e_1(B,Y,Z)$ & $e_2(Y)$ & $q(Y,Z)$ & $p(X,Z)$.

$(S2)$: $q(Y,Z)$:- $e_1(A,Y,Z)$.

$(S3)$: $q(Y,Z)$:- $s(X,Y)$ & $p(Y,Z)$.

If a full sips is used in rule $S1$, the subgoals $e_1(B,Y,Z)$ and $e_2(Y)$ bind variable Y and Z before subgoal $q(Y,Z)$ is evaluated. The subgoal $q(Y,Z)$ then provides an existential test for the assigned (Y,Z) values. However, rule $S2$ defines $q(Y,Z)$ to be true whenever $e_1(B,Y,Z)$ is true, making the existential test redundant. The subgoal $q(Y,Z)$ can thus be elinated from rule $S1$ to derive an equivalent program T:

$(T1)$: $s(X,Y)$:- $e_1(B,Y,Z)$ & $e_2(Y)$ & $p(X,Z)$.

$(T2)$: $q(Y,Z)$:- $e_1(A,Y,Z)$.

$(T3)$: $q(Y,Z)$:- $s(X,Y)$ & $p(Y,Z)$.

Note that eliminating subgoal $q(Y,Z)$ transformed a recursive program S into an equivalent nonrecursive program T. □

We now describe a general algorithm to eliminate redundant subgoals as in the example above. The algorithm uses the idea of containment mappings [Ull89]. However, we note that the conjunctive query containment techniques discussed in [Ull89] cannot eliminate subgoal $q(Y,Z)$ from rule $S1$ in the example above.

Definition 4.1 Minimal Cover: Consider the rule r:

(r): s :- g_1 & ... & g_k & q & g_{k+1} & ... & g_n.

with $q, g_1, \ldots, g_n$ as subgoals. Let the function VAR return the variables that occur as arguments in a given set of subgoals. Let $\mathcal{P} \subseteq \{g_1, \ldots, g_n\}$ be the set of subgoals that pass information sideways into subgoal q according to a given sips. A set of subgoals $\mathcal{C} \subseteq \mathcal{P}$ is said to be a *cover* for subgoal q if $\mathsf{VAR}(q) \subseteq \mathsf{VAR}(\mathcal{C})$. Subgoal q is said to be *covered* by set $\mathcal{C}$. A set $\mathcal{C}$ is said to be a *minimal cover* for q if it is a cover for q and if no subset of $\mathcal{C}$ is a cover for q. □

EXAMPLE 4.2 Consider the rule $S1$ from Example 4.1. Using a full sips, the set $\{e_1(B,Y,Z)\}$ is a minimal cover for the subgoal $q(Y,Z)$. The set $\{e_1(B,Y,Z), e_2(Y)\}$ is a cover for subgoal $q(Y,Z)$, but it is not minimal

Consider the following rule u:

(u): $s(X,Y)$:- $e_1(X,Z)$ & $e_2(A,C)$ & $q(A,Z)$ & $e_3(A,B)$ & $p(B,C,Y)$.

Using full sips, the set $\{e_1(X,Z), e_2(A,C)\}$ is a minimal cover for the subgoal $q(A,Z)$. The set $\{e_1(X,Z), e_3(A,B)\}$ is not a minimal cover since e_3 does not pass information into q under a full sips. □

The subgoals in a cover determine a set of bindings for the variables in the covered subgoal. If we can guarantee that the covered subgoal will be true for each of these bindings, then the covered subgoal is redundant.

Definition 4.2 Covered Subgoal Elimination: Given a program with the following pair of rules:

$(r1)$: $p(t_1, \ldots, t_n)$:- S_1 & $q(u_1, \ldots, u_m)$ & S_2.

$(r2)$: $q(v_1, \ldots, v_m)$:- S_3.

where S_1, S_2, and S_3 are conjunctions of subgoals, consider the following two conditions:

1. There exists a minimal cover S_0 for the subgoal $q(u_1, \ldots, u_m)$ in rule $r1$.
2. Let rule $r3$ be defined as follows:

   $(r3)$: $q(u_1, \ldots, u_m)$:- S_0.

   Then, there is a containment mapping [Ull89] from rule $r2$ to rule $r3$. That is, if rules $r2$ and $r3$ are viewed as conjunctive queries, the conjunctive query $r3$ is contained in the conjunctive query $r2$.

If Conditions 1 and 2 are satisfied, then subgoal $q(u_1, \ldots, u_m)$ can be eliminated from rule $r1$. □

EXAMPLE 4.3 Consider program S introduced in Example 4.1. Using a full sips in rule $S1$, the set $\{e_1(B,Y,Z)\}$ is a minimal cover for the subgoal $q(Y,Z)$. Also, there exists a containment mapping from rule

$q(Y,Z)$:- $e_1(A,Y,Z)$.

to the rule

$q(Y,Z)$:- $e_1(B,Y,Z)$.

The containment mapping is $< Z \rightarrow Z, Y \rightarrow Y, A \rightarrow B >$. Subgoal $q(Y,Z)$ is thus redundant, and can be eliminated from rule $S1$. □

Theorem 4.1 *Let CP be the program obtained by eliminating a redundant subgoal from a given program P using covered subgoal elimination. Programs P and CP are equivalent.* □

Covered subgoal elimination can be used in programs where duplicates are important. Using duplicate semantics for programs with `set` and `multiset` predicates (such as SQL programs) as defined in [MPR90], we have the following result:

Theorem 4.2 *Let CP be the program obtained by eliminating redundant subgoals for* `set` *predicates from a given program P using covered subgoal elimination. Programs P and CP are duplicate equivalent.* □

5 MST/CSE

We use covered subgoal elimination to develop an algorithm for doing the magic-sets transformation on a nonrecursive program so that the transformed program is also nonrecursive. The algorithm consists of two phases: (1) Apply the regular magic-sets transformation (MST, Definition 1.3), possibly deriving a recursive program, and (2) Use covered subgoal elimination to remove recursion. Hence the name MST/CSE.

Given a nonrecursive program P, any recursion in the magic program MP obtained by MST is due to a predicate in a higher stratum passing information into a predicate in a lower stratum through magic predicates (the strata of predicates are determined from the dependency graph of the original program). However, such information passing is also reflected by a cycle in the information passing graph, $\mathsf{IPG}(P)$, of the original program P. MST/CSE eliminates recursion in the magic program by a sequence of unfolding [TS84] and CSE steps that correspond to breaking of cycles in the information passing graph $\mathsf{IPG}(P)$.

We explain the process of breaking cycles in an information passing graph. Let C be a cycle, and let e be an edge in the cycle of the form $p \rightarrow q$ or $p \Rightarrow mq$, where p and q are non-magic nodes, and mq is a magic node. By Lemma 2.1, every cycle must have at least one non-magic node, so such an edge e exists. We can modify the magic program MP in a manner that corresponds to breaking the edge e. If edge e is of the form $p \rightarrow q$, simply *unfold* [TS84] every occurrence of subgoal p in every rule for predicate q in the magic program MP. Unfolding a subgoal p involves replacing the subgoal p by the body of the rules defining predicate p. If there are multiple rules that define predicate p, then unfolding an instance of p in a rule r results in multiple rules, one from each rule for p. Since p is a non-magic node, rules for predicate p may have the magic predicate m_p as a subgoal, which then appears in each unfolded rule. By Theorem 5.1 given below, the magic predicate m_p is redundant in the unfolded rules, and can be eliminated by CSE.

Theorem 5.1 *Given a program P, let MP be the program obtained by the magic-sets transformation of P. Let p be a non-magic predicate, and let r be a rule in program MP with predicate p as a subgoal. If subgoal p is unfolded in rule r, the resulting instance of the subgoal m_p in the unfolded rule is redundant.* □

Unfolding the p subgoals and eliminating the magic subgoal m_p from the unfolded rules corresponds to replacing the edge $e = p \rightarrow q$ in $\mathsf{IPG}(P)$ with edges from subgoals in the rules for p to q, except that the edge from m_p to q is not added. As a result the cycle C containing the edge e may get broken.

EXAMPLE 5.1 Consider the rules $MP2$, $MP3$, $MP7$, and $MP8$ from program MP derived in Example 1.1.

$(MP2)$: $s(X,Y)$:- $m\_s(X)$ & $p(X)$ & $q(X,Y)$.

$(MP3)$: $p(X)$:- $m\_p(X)$ & $e_1(X,Y)$ & $e_2(Y)$.

$(MP7)$: $m\_p(X)$:- $m\_t(X)$ & $s(X,Y)$.

$(MP8)$: $m\_p(X)$:- $m\_s(X)$.

Rule $MP2$ corresponds to the edge $p \rightarrow s$ in $\mathsf{IPG}(P)$. Rules $MP3$ and $MP7$ correspond to edges $mp \Rightarrow p$ and $s \Rightarrow mp$ respectively. These three edges together form the cycle $C = (s \Rightarrow mp \Rightarrow p \rightarrow s)$. Let us break the edge $p \rightarrow s$. We start by unfolding the p subgoal in rule $MP2$, that is we replace the p subgoal by the body of rule $MP3$:

$(MP2_u)$: $s(X,Y)$:- $m\_s(X)$ & $m\_p(X)$ & $e_1(X,Z)$ & $e_2(Z)$ & $q(X,Y)$.

In the presence of rule $MP8$, subgoal m_p in the above rule $MP2_u$ can be eliminated by CSE to obtain rule $MP2'$:

$(MP2')$: $s(X,Y)$:- $m\_s(X)$ & $e_1(X,Z)$ & $e_2(Z)$ & $q(X,Y)$.

Replacing rule $MP2$ by rule $MP2'$ corresponds to replacing the edge $p \rightarrow s$ in $\mathsf{IPG}(P)$ by the edges $e_1 \rightarrow s$ and $e_2 \rightarrow s$. □

In Example 5.1, breaking the edge $p \rightarrow s$ also breaks the cycle C, making program MP nonrecursive. However, breaking an edge of a cycle does not always break the cycle, as the following example illustrates.

EXAMPLE 5.2 Consider the rules and the cycle C introduced in the above Example 5.1. Let us now break the edge $s \Rightarrow mp$ in cycle C. We unfold the s subgoal in rule $MP7$ to derive rule $MP7_u$:

$(MP7_u)$: $m\_p(X)$:- $m\_t(X)$ & $m\_s(X)$ & $p(X)$ & $q(X,Y)$.

Algorithm 5.1
Input: A nonrecursive program P.
$\mathsf{IPG}(P)$, the information passing graph of program P.

Output: A nonrecursive program equivalent to the magic-sets transformation of program P.

Method

```
% Phase 1: Regular Magic-sets transformation
Do magic-sets transformation on program P to get program MP;
Let 𝒞 = the set of simple cycles in IPG(P);
if (𝒞 = ∅)                                              % IPG(P) does not have cycles
  Output Program MP;                                                    % We are done
else                                                        % Program MP is recursive
 % Phase 2: Remove cycles in program MP
 {while (there exists a cycle in set 𝒞)                       % Remove cycles one by one
      % Step 2.A: Select a cycle and an edge within the cycle
      {Choose a cycle C that has the least CSN amongst all cycles in 𝒞;
       Choose a predicate p that has the least stratum number amongst all predicates in cycle in C;
       Let e be the outedge from node p in cycle C;

       % Step 2.B: Identify the rules of program MP to be modified.
       ℛ = ∅;                                               % ℛ = set of rules to be modified
       if (edge e is of the form (p → q))                                % a dependency edge
          s = q;                   % s is the predicate corresponding to node q in program MP
       if (edge e is of the form (p ⇒ mq))                             % an information edge
          s = m_q;                    % magic node mq corresponds to the magic predicate m_q
       for all rules r ∈ program MP
           if ((s = head(r)) and (p ∈ body(r)))
              ℛ = ℛ ∪ {r};                                      % Rule r needs to be modified

       % Step 2.C: Modify rules by unfolding and CSE
       for all (rules r ∈ ℛ)
           {Unfold all instances of predicate p in the body of rule r;
            Eliminate subgoal m_p from the unfolded rules;                % Application of CSE
           }
       𝒞 = 𝒞 − {C};                                          % Delete cycle C from the set 𝒞
      }
  Output the modified program MP;                                            % We are done
 }
```

◇

Figure 4: The MST/CSE algorithm

In the presence of rule $MP6$ (see Example 1.1), subgoal m_s in the above rule $MP7_u$ can be eliminated by CSE to obtain rule $MP7'$:

$$(MP7'): \quad m\_p(X) \text{ :- } m\_t(X) \ \& \ p(X) \ \& \ q(X, Y).$$

However, the corresponding changes in $\mathsf{IPG}(P)$ do not break cycle C; Cycle C is converted to the cycle $p \Rightarrow mp \Rightarrow p$. □

Thus, the selection of an edge to be broken is important in breaking a cycle. In addition, if the information passing graph has multiple cycles, the order in which cycles are broken is also important.

The MST/CSE algorithm (Algorithm 5.1) is shown in Figure 4. Algorithm 5.1 breaks cycles one by one, in increasing order of their cycle stratum numbers. Each cycle is broken by breaking a single edge: the edge e of the form $p \rightarrow q$ or $p \Rightarrow mq$ where node p has the least stratum number amongst all non-magic nodes in the cycle. We can show (proof to Theorem 5.2, [GM92]) that breaking edges in the above order breaks cycles and does not create alternative cycles.

Looking back at the cycle $C = (s \Rightarrow mp \Rightarrow p \rightarrow s)$ in Example 5.1, node p is in stratum 1, and node s is in stratum 2. Algorithm 5.1 breaks the edge $p \rightarrow s$ in preference to the edge $s \Rightarrow mp$.

EXAMPLE 5.3 We will apply Algorithm 5.1 to program P introduced in Example 1.1. We will use the information passing graph shown in Figure 2 (Page 4) and the cycles identified in Table 1 (Page 5).

Phase 1: *Regular Magic-sets transformation*
Derive the magic program MP (Example 1.1)
Cycles in $\mathsf{IPG}(P)$: $\mathcal{C} = \{C1, C2\}$ (Table 1)

Phase 2: *Remove cycles*
Iteration 1.
Step 2.A
Cycles with CSN=1: $C1 = \{p, mq, q, s, mp, p\}$,
$C2 = \{p, s, mp, p\}$
Cycle selected: $C1$
(one of the cycles with CSN=1)
Predicates in lowest stratum: p, q
Predicate selected for unfolding: p
(one of the eligible predicates)
Outgoing Edge: $p \Rightarrow mq$
Step 2.B
Rules to be modified: $\mathcal{R} = \{MP9\}$
Step 2.C
Unfold predicate p in rule $MP9$

$$m\_q(X) :- m\_s(X) \ \&\ m\_p(X) \ \&\ e_1(X,Z) \ \&\ e_2(Z).$$

Eliminate subgoal m_p to get rule $MP9'$

$$(MP9'): \quad m\_q(X) :- m\_s(X) \ \&\ e_1(X,Z) \ \&\ e_2(Z).$$

Cycles broken: $C1 = \{p, mq, q, s, mp, p\}$

Iteration 2.
Step 2.A
Cycles with CSN=1: $C2 = \{p, s, mp, p\}$
Cycle selected: $C2$
Predicates in the lowest stratum: p
Predicate selected for unfolding: p
Outgoing edge: $p \rightarrow s$
Step 2.B
Rules to be modified: $\mathcal{R} = \{MP2\}$
Step 2.C
Unfold predicate p in rule $MP2$

$$s(X,Y) :- m\_s(X) \ \&\ m\_p(X) \ \&\ e_1(X,Z) \ \&\ e_2(Z) \ \&\ q(X,Y).$$

Figure 5: Partial Information Passing Graphs of programs S and T in Example 6.1

Eliminate subgoal m_p to get rule $MP2'$

$$(MP2'): \quad s(X,Y) :- m\_s(X) \ \&\ e_1(X,Z) \ \&\ e_2(Z) \ \&\ q(X,Y).$$

Cycles broken: $C2 = \{p, s, mp, p\}$

The modified program MP output by Algorithm 5.1:

$(MP1)$: $t(X) :- m\_t(X) \ \&\ s(X,Y) \ \&\ p(Y).$

$(MP2')$: $s(X,Y) :- m\_s(X) \ \&\ e_1(X,Z) \ \&\ e_2(Z) \ \&\ q(X,Y).$

$(MP3)$: $p(X) :- m\_p(X) \ \&\ e_1(X,Z) \ \&\ e_2(Z).$

$(MP4)$: $q(X,Y) :- m\_q(X) \ \&\ e_3(X,Z) \ \&\ e_4(Z,Y).$

$(MP5)$: $m\_t(0).$

$(MP6)$: $m\_s(X) :- m\_t(X).$

$(MP7)$: $m\_p(Y) :- m\_t(X) \ \&\ s(X,Y).$

$(MP8)$: $m\_p(X) :- m\_s(X).$

$(MP9')$: $m\_q(X) :- m\_s(X) \ \&\ e_1(X,Z) \ \&\ e_2(Z).$

□

Theorem 5.2 *Algorithm 5.1 terminates, and produces a nonrecursive program equivalent to the magic-sets transformation of the original program.* □

6 RLCS/MST

As discussed in Section 3, eliminating all common subexpressions from a program ensures that its magic-sets transformation is nonrecursive. But often it is not necessary to eliminate all common subexpressions, as Example 6.1 below illustrates. We develop another two phase algorithm for doing magic-sets transformation on nonrecursive programs that (1) replicates some of the common subexpressions, and (2) applies the regular magic-sets transformation (MST). Enough common subexpressions are eliminated during the first phase to guaranty that MST in the second phase does not derive a recursive program. The algorithm is called RLCS/MST, short for *Replicate Limited Common Subexpressions and apply Magic-Sets Transformation.*

EXAMPLE 6.1 Consider the following program S:

(Q): ?- s(a,Y).

$(S1)$: $s(X,Y)$:- $p(X,Z)$ & $p(Z,Y)$.

$(S2)$: $t(X,Y)$:- $e_1(X,Z)$ & $p(Z,Y)$.

$(S3)$: $p(X,Y)$:- $e_2(X,Y)$.

The information passing graph of program S, $\mathsf{IPG}(S)$ is shown in Figure 5. We have omitted some of the magic nodes from the figure for simplicity.

Predicate p appears in three subgoals in Program S. RACS/MST replicates predicate p twice to create three copies of p, one for each subgoal, thereby eliminating all common subexpressions. However, we do not need to eliminate all common subexpressions. Replicating predicate p once to create two copies of p, say p and p_1, and replacing the second subgoal in rule $S1$ by predicate p_1, we derive the following program T:

$(T1)$: $s(X,Y)$:- $p(X,Z)$ & $p_1(Z,Y)$.

$(T2)$: $t(X,Y)$:- $e_1(X,Z)$ & $p(Z,Y)$.

$(T3)$: $p(X,Y)$:- $e_2(X,Y)$.

$(T4)$: $p_1(X,Y)$:- $e_2(X,Y)$.

Predicate p_1 is defined by rule $T4$ that is identical to rule $S3$ defining predicate p. The information passing graph of program T (Figure 5) is acyclic; Hence by Theorem 2.1, the magic-sets transformation of program T is nonrecursive. □

We now define some concepts needed to develop the RLCS/MST algorithm.

Definition 6.1 Firm Node: Given a program P, consider the node for predicate p and a cycle C in $\mathsf{IPG}(P)$. A non-magic node p in cycle C is firm with respect to (wrt) cycle C if one of the following two conditions is true.

- The inedge into node p in cycle C is a **D** edge, or
- The inedge into node p in cycle C is an **I** edge, and the following condition holds: Let e_o be the outedge from node p in cycle C and let e_i be the inedge into node mp in cycle C. Let $\mathcal{I}$ and $\mathcal{O}$ be defined as follows:

  $\mathcal{I} = \{< r,\ d > \ \mid\ < r,\ s,\ d > \in \mathsf{label}(e_i)\}$.
  $\mathcal{O} = \{< r,\ s > \ \mid\ < r,\ s,\ d > \in \mathsf{label}(e_o)\}$.
  $\mathcal{I} = \mathcal{O}$ and each has only one element.

□

EXAMPLE 6.2 (Firm Nodes) Consider the following rules that define program L:

(Q): ?- $t(0,Y)$.
$(L1)$: $t(X,Y)$:- $p(X,Y)$ & $q(Y,Z)$ & $s(Z,A)$.
$(L2)$: $t(X,Y)$:- $s(X,Z)$ & $p(Z,Y)$.
$(L3)$: $t(X,Y)$:- $q(X,Z)$ & $e_1(Y,Z)$.

Node q in rule $L1$ is firm wrt the cycle C: $p \Rightarrow mq \Rightarrow q \Rightarrow ms \Rightarrow s \Rightarrow mp \Rightarrow p$. The label on edge $p \Rightarrow mq$ is $< L1, 1, 2 >$ and on edge $q \Rightarrow ms$ is $< L1, 2, 3 >$. q is firm with respect to the cycle because both labels correspond to the same instance of predicate q i.e. $\mathcal{I} = \mathcal{O} = \{< L1, 2 >\}$. Node s on the other hand is not firm wrt C because the edge $q \Rightarrow ms$ corresponds to the instance of s in rule $L1$ and the edge $s \Rightarrow mp$ corresponds to s in rule $L2$.
□

Theorem 6.1 *Say the body of a rule R in program P, has multiple instances of a predicate p. If there is sideways information passing from one instance of p to another instance of p, then $\mathsf{IPG}(P)$ contains the cycle $p \Rightarrow mp \Rightarrow p$. In addition, node p is not firm with respect to cycle $p \Rightarrow mp \Rightarrow p$.* □

Theorem 6.1 is used to optimize Algorithm 6.1. Step 1 of the algorithm eliminates all cycles that are caused by repeated occurences of predicates within any rule. The elimination is done by replacing all but one occurences of the repeated predicate. The IPG of the modified program is then used to identify the remaining cycles in the program. A cycle is broken by eliminating the outedges from a node that is not firm with respect to the cycle.

EXAMPLE 6.3 Consider program L from Example 6.2. Note that node q was firm with respect to cycle C. Say we replace the predicate q by a new predicate q_1 in rule $L1$ to break the edge $q \Rightarrow ms$. In the cycle C, the edges $p \Rightarrow mq$ and $q \Rightarrow ms$ get replaced by edges $p \Rightarrow mq_1$ and $q_1 \Rightarrow ms$ respectively. As a consequence cycle C gets replaced by the cycle $p \Rightarrow mq_1 \Rightarrow q_1 \Rightarrow ms \Rightarrow s \Rightarrow mp \Rightarrow p$.

Instead let predicate s be replaced by s_1 in rule $L2$. In the IPG, the outedge $s \Rightarrow mp$ is replaced by edge $s_1 \Rightarrow mp$. The inedge $q \Rightarrow ms$ does not change and consequently cycle C is broken. □

Replicating the predicate for a node that is firm with respect to a cycle does not help (Example 6.3).

Algorithm 6.1
Input: A nonrecursive program P.
$\mathsf{IPG}(P)$, the information passing graph of program P.

Output: A program P' equivalent to P such that magic-sets transformation of P' is nonrecursive.

Method
% *Step 1: Eliminate multiple occurences of predicates from rules*
$\mathcal{P}$ = Rules of program P arranged in order of increasing RSN.
for each rule $r \in \mathcal{P}$ in order
$\mathcal{O} = \{\ p \mid \mathsf{body}(r)$ has multiple instances of p and atleast one instance passes information sideways to another instance of p in r.$\}$
for each predicate $p \in \mathcal{O}$
for each instance of $p \in \mathsf{body}(r)$ except the first
Replace p by a new predicate p_i. % *Eliminate repeated occurences of p from r*
Define p_i by the same rules that define predicate p.
Put the modified rule r and the definitions for all new predicates in program P'.
Construct the IPG of the program P'.
Let $\mathcal{C}$ = the set of simple cycles in $\mathsf{IPG}(P')$.
if $(\mathcal{C} = \emptyset)$ % $\mathsf{IPG}(P')$ *does not have cycles*
Output Program P'. % *We are done*
else % *Program P' needs further subexpression elimination*
% *Step 2: Remove cycles from program P' by processing predicates in increasing order of their* SN*s*
Let Q be the set of original derived predicates in P. % *Magic predicates and new p_i's are not considered*
While $(Q \neq \emptyset)$
% *Step 2.A*
{ Choose a predicate $p \in Q$ amongst the predicates with the least stratum number.
While there exists a cycle C in $\mathsf{IPG}(P')$ such that p is not firm wrt C
% *Step 2.B: Try to break cycle C by replacing p*
{ Let $e \in \mathsf{edges}(C)$ be the outgoing edge from node p.
Use $\mathsf{label}(e)$ to identify the relevant instances of predicate p.
for each relevant instance of p % *Make the changes in program P'*
% *Step 2.C*
Replace p by a new predicate p_i. % *Common subexpression elimination*
Define p_i by the same rules that define predicate p.
Recompute $\mathsf{IPG}(P‘)$.
}
$Q = Q - \{p\}$ % *Remove predicate p from set Q*
}
$\diamond$

Figure 6: The RLCS/MST algorithm

In general it is useless to replicate a predicate that is firm with respect to all cycles.

Lemma 6.1 *Every cycle C has at least one predicate that is not firm with respect to C.* $\square$

Algorithm 6.1 breaks cycles by replicating predicates corresponding to nodes that are not firm with respect to some cycle. Lemma 6.1 guarantees that every cycle has a non-firm node and gets broken eventually. The algorithm first replicates non-firm nodes whose predicates have the smallest stratum numbers in the program. Predicates in higher strata are replicated only when predicates with smaller stratum numbers are firm with respect to all cycles. For instance in the program L from Example 6.2, if predicate p had a smaller stratum number than s, then the algorithm would first replicate p to break the cycle C.

Common subexpression elimination introduces rules that define the new predicate. The subgoals in the new rules introduce dependencies that could add more cycles to the program. Processing predicates in order of their stratum numbers guarantees that we do not duplicate cycles while replication. However ordered replication may sometimes lead to inefficiency. Consider two cycles $C1$ and $C2$ that could both be broken by replicating predicate s that does not have the smallest stratum number in the cycles. However, Algorithm 6.1 would replicate predicates with stratum numbers less than s in cycles $C1$ and $C2$. As a consequence more than one predicate may be replicated even though replicating just s would have been sufficient.

EXAMPLE 6.4 Algorithm 6.1 is now applied to program P of Example 1.1. Note that restricted common subexpression elimination is done on the rules of the original program P.

Step 1: Program P remains unchanged.
Step 2:
Iteration 1.
Step 2.A
Predicate chosen: p (one of the predicates with smallest stratum number)
C_p : $\{C1 = \{p, mq, q, s, mp, p\},\ C2 = \{p, s, mp, p\}\}$
p is not firm wrt either $C1$ and $C2$
Step 2.B
Cycle selected: $C1$ (one of the eligible cycles)
Edge to be broken: $p \Rightarrow mq$
outgoing edge from p that is part of $C1$
Relevant instances of p: p in Rule $P2$
Step 2.C
Replace p in rule $P2$ to get:
$(P2')$: $s(X,Y)$:- $p_1(X)$ & $q(X,Y)$.
Cycles broken: $C1$, $C2$
because edge $p \rightarrow s$ is also eliminated.

The transformed program is the same as program P_h derived in Example 3.1. (RACS/MST and RLCS/MST have identical behavior since there is only one common subexpression in program P, and it must be replicated to avoid recursion.) □

Theorem 6.2 *Algorithm 6.1 terminates, and derives a program whose magic-sets transformation is nonrecursive and equivalent to the magic-sets transformation of the original program.* □

7 Conclusions

We have solved an important problem: how to use the magic-sets transformation in database systems that do not support recursion.

We have developed magic-sets transformation algorithms that guarantee a nonrecursive output when given a nonrecursive program as input. These algorithms are useful because they make the efficient magic-sets transformation (MST) usable by current systems most of which do not support recursion. The performance improvements obtainable by MST would be lost to nonrecursive systems if the recursion introduced by MST could not be eliminated.

We have discussed two nonrecursive magic-sets transformation algorithms in detail, MST/CSE and RLCS/MST. Both the algorithms work by selective replication of common subexpressions: MST/CSE replicates by unfolding, and RLCS/MST replicates by copying predicates. The amount of replication can differ on some examples, and we are currently studying the algorithms to better understand the difference. Both algorithms can preserve duplicate semantics [MPR90], and can therefore be used in SQL systems. However, unfolding is not always possible in SQL queries (as when doing grouping), and RLCS/MST may be be the only applicable algorithm on such queries. The MST/CSE and RLCS/MST algorithms both work with adorned programs; we presented them using unadorned programs for simplicity.

In addition to the above two algorithms, we have also developed a 1-Phase MST/CSE algorithm [GM92]. There are a few problems that need further work. In what order should one break the cycles in the information passing graph? Within a cycle, how should one select common subexpressions to be replicated so as to minimize the amount of replication? Can one always select a cycle order and a common subexpression within each cycle such that MST/CSE and RLCS/MST do identical replication? What is the performance penalty due to replication? The RLCS/MST algorithm replicates fewer common subexpressions than RACS/MST, but RLCS/MST can still replicate common subexpressions unnecessarily [GM92]. We would like to develop an algorithm that replicates a common subexpression only if the replication is necessary to avoid recursion.

A second contribution of this paper is the *covered subgoal elimination* (CSE) technique to optimize rule evaluation. CSE is related to conjunctive query minimization and minimization of unions of conjunctive queries ([Ull89], Chapter 14) in that they all use conjunctive query containment as a tool. CSE checks redundancy of subgoals by looking at other rules in the program. Redundant subgoals are then eliminated, and a recursive query may thus become nonrecursive.

With the performance advantage of the magic-sets transformation on nonrecursive queries well established [MFPR90], we expect this paper to have a real impact on how query optimization is done in commercial database systems.

8 Acknowledgements

We thank Hamid Pirahesh and Jeff Ullman for discussions, and Oded Shmueli for valuable comments on a draft of the paper.

References

[BR87] Catriel Beeri and Raghu Ramakrishnan. On the power of magic. In *Proceedings of the Sixth Symposium on Principles of Database Systems (PODS)*, pages 269–283, San Diego, CA, March 1987. ACM SIGACT-SIGMOD-SIGART.

[GM92] Ashish Gupta and Inderpal Singh Mumick. Magic-sets transformation in non-recursive systems. Technical report, Stanford University, Stanford, CA 94305, USA, 1992. To be published.

[ISO90] ISO_ANSI. Iso-ansi working draft: Database language sql2 and sql3; x3h2; iso/iec jtc1/sc21/wg3, 1990.

[MFPR90] Inderpal Singh Mumick, Sheldon J. Finkelstein, Hamid Pirahesh, and Raghu Ramakrishnan. Magic is relevant. In *Proceedings of ACM SIGMOD 1990 International Conference on Management of Data*, pages 247–258, Atlantic City, NJ, May 23-25 1990.

[MPR90] Inderpal Singh Mumick, Hamid Pirahesh, and Raghu Ramakrishnan. The magic of duplicates and aggregates. In *Proceedings of the Sixteenth International Conference on Very Large Databases (VLDB)*, pages 264–277, Brisbane, Australia, August 13-16 1990.

[Mum91] Inderpal Singh Mumick. *Query Optimization in Deductive and Relational Databases*. PhD thesis, Stanford University, Stanford, CA 94305, USA, December 1991.

[Nau86] Jeffrey F. Naughton. Data independent recursion in deductive databases. In *Proceedings of the Fifth Symposium on Principles of Database Systems (PODS)*, pages 267–279, Cambridge, MA, 1986.

[TS84] H. Tamaki and T. Sato. Unfold/fold transformations in logic programs. In S. A. Tarnlund, editor, *Proc. of the Second International Conference on Logic Programming*, pages 127–138, Uppsala, Sweden, 1984.

[Ull89] Jeffrey D. Ullman. *Principles of Database and Knowledge-Base Systems, Volumes 1 and 2*. Computer Science Press, 1989.