Magic Conditions

Presented at PODS 1990

Inderpal Singh Mumick[*]
Stanford University

Sheldon J. Finkelstein[†]
IBM Almaden Research Center

Hamid Pirahesh
IBM Almaden Research Center

Raghu Ramakrishnan[‡]
University of Wisconsin at Madison

Abstract

Much recent work has focussed on the bottom-up evaluation of Datalog programs. One approach, called magic-sets, is based on rewriting a logic program so that bottom-up fixpoint evaluation of the program avoids generation of irrelevant facts ([BMSU86, BR87, Ram88]). It is widely believed that the principal application of the magic-sets technique is to restrict computation in recursive queries using equijoin predicates. We extend the magic-sets transformation to use predicates other than equality ($X > 10$, for example). This Extended Magic-Set technique has practical utility in "real" relational databases, not only for recursive queries, but for non-recursive queries as well; in ([MFPR90]) we use the results in this paper and those in [MPR90] to define a magic-set transformation for relational databases supporting SQL and its extensions, going on to describe an implementation of magic in Starburst ([HFLP89]). We also give preliminary performance measurements.

In extending magic-sets, we describe a natural generalization of the common class of bound (b) and free (f) adornments. We also present a formalism to compare adornment classes.

[*]Part of this work was done at the IBM Almaden Research Center. Work at Stanford was supported by an NSF grant IRI-87-22886, an Air Force grant AFOSR-88-0266, and a grant of IBM Corporation.

[†]Author's current affiliation: Tandem Computers

[‡]This work was done while the author was visiting IBM Almaden Research Center

1 Introduction

The idea behind the magic-sets technique is to compute a set of auxiliary ("magic") predicates for the bindings on the goals in a program's rules . These rules are rewritten using the magic predicates so that irrelevant tuples are not generated. The rewriting is guided by a choice of *sideways information passing strategy*, or SIPS[1] for each rule, that dictates how information is to be passed between subgoals in the rule. The rewriting algorithm is a two-step transformation. First an *adorned* version, P^{ad}, of program P is produced. Predicates in P^{ad} are annotated with information about the values that can appear in different argument positions. The annotations are called adornments. In the second step of the rewrite, magic predicates are introduced into P^{ad}.

Previous work on magic-sets has used a bf adornment pattern that distinguishes bound (b) and free (f) argument positions. The interpretation of bound and free varies greatly: some treatments consider an argument position in a body literal of a rule to be bound if the variable in it is bound to a constant[2] in all goals generated from this literal (e.g., [BMSU86, BR87]); others consider the argument bound if the variable in it is potentially restricted — possibly even free — in goals generated from this literal (e.g., [Ram88]). In this paper, we introduce a new class of adornment patterns by defining a c adornment that describes selections involving arithmetic inequalities (*conditions*), or, more generally, any built-in predicate. Such conditions are widely used in practical database queries. Examples are: salary greater than 50K (a condition on one subgoal), or last year's sales more than this year's sales (a condition between two subgoals). For large databases, magic transformations using bindings obtained from

[1]Note that SIPS is an acronym for both, *a* sideways information passing strategy, and, *many* sideways information passing strategies.

[2]More generally, to a ground term.

such conditions may yield orders-of-magnitude performance improvements. We refer to the new class of adornments patterns as the bcf class. SIPS are used to specify how information is passed in the rule bodies, and we extend the definition of SIPS to allow us to specify how conditions, in addition to bindings, are passed sideways.

We show how the magic-sets rewriting algorithm can be modified to support propagation of arithmetic conditions. In related work, the Magic Templates algorithm ([Ram88]) also permits conditions to be propagated, but at the cost of maintaining non-ground tuples with associated conditions on the free variables. The method we present, Ground Magic-sets Transformation, involves only ordinary ground tuples. Kemp et al. ([BKMR89]) have presented another approach to refining the Magic rewriting for propagating arithmetic conditions while dealing only with ground tuples. However, their approach is quite different from ours; in particular, it does not exploit the descriptive power of condition adornments.

The following example motivates the problem of propagating conditions, and illustrates our solution.

EXAMPLE 1.1 (Motivation): Consider the Same Generation program P:[3]

$(P1)$: $sg(X,Y)$:- $\texttt{flat}(X,Y)$.
$(P2)$: $sg(X,Y)$:- $\texttt{up}(X,U)$ & $sg(U,V)$ & $\texttt{down}(V,Y)$.

We use the bf adornment pattern of [BMSU86, BR87]. For the query

(Q): ?- $X > 10$ & $sg(X,Y)$.

the condition on X cannot be captured by the b adornment. The f adornment does not describe the condition, and in effect discards it; however, b cannot be used (since X is not bound), so the adorned query is sg^{ff}. Thus, the full sg relation will be computed before the restriction is applied to X.

It would be better to solve this query by pushing the selection condition into the recursion in the following manner:

1. Find all U such that $X > 10$ & $\texttt{up}(X,U)$. Call this set T.
2. Solve the sg^{bf} query with T as the set of bindings for the first argument. Denote the set of resulting sg tuples as S.
3. Compute the answer to the query by two rule evaluations: (i) Apply the non-recursive rule $P1$ with the additional condition $X > 10$, and (ii) Apply a non-recursive rule corresponding to $P2$, with the set S replacing the sg literal, and with the additional condition $X > 10$.

[3] We often use a letter, such as P, to represent a program comprised of rules $P1, P2, \ldots$

Using the c adornment (formally described in Section 3.1) leads to the following adorned program that suggests the computation described above.

(Qa): ?- $X > 10$ & $sg^{cf}(X,Y)$.

$(A1)$: $sg^{cf}(X,Y)$:- $\texttt{flat}^{cf}(X,Y)$.
$(A2)$: $sg^{cf}(X,Y)$:- $\texttt{up}^{cf}(X,U)$ & $sg^{bf}(U,V)$ & $\texttt{down}^{bf}(V,Y)$.

$(A3)$: $sg^{bf}(X,Y)$:- $\texttt{flat}^{bf}(X,Y)$.
$(A4)$: $sg^{bf}(X,Y)$:- $\texttt{up}^{bf}(X,U)$ & $sg^{bf}(U,V)$ & $\texttt{down}^{bf}(V,Y)$.

The magic-templates transformation yields the program M:

$(M1)$: $sg^{cf}(X,Y)$:- $m\_sg^{cf}(X)$ & $\texttt{flat}^{cf}(X,Y)$.
$(M2)$: $sg^{cf}(X,Y)$:- $m\_sg^{cf}(X)$ & $\texttt{up}^{cf}(X,U)$ & $sg^{bf}(U,V)$ & $\texttt{down}^{bf}(V,Y)$.

$(M3)$: $sg^{bf}(X,Y)$:- $m\_sg^{bf}(X)$ & $\texttt{flat}^{bf}(X,Y)$.
$(M4)$: $sg^{bf}(X,Y)$:- $m\_sg^{bf}(X)$ & $\texttt{up}^{bf}(X,U)$ & $sg^{bf}(U,V)$ & $\texttt{down}^{bf}(V,Y)$.

$(M5)$: $m\_sg^{cf}(X)$:- $X > 10$.

$(M6)$: $m\_sg^{bf}(U)$:- $m\_sg^{cf}(X)$ & $\texttt{up}^{cf}(X,U)$.
$(M7)$: $m\_sg^{bf}(U)$:- $m\_sg^{bf}(X)$ & $\texttt{up}^{bf}(X,U)$.

Rule $(M6)$ computes the set T of Step 1 above.

However, rule $M5$ is not range-restricted, and we cannot compute the magic set of sg^{cf} as a set of ground tuples. If we look closely, m_sg^{cf} is only used in rules $M1$ and $M2$[4] where the magic values are *grounded* by non-recursive subgoals. Our ground magic-sets transformation computes only the magic values that would be useful in $M1$ and $M2$ by *grounding* the magic set $X > 10$ as follows (with $M5$ replaced by $M5'$ and $M5''$):

$(M5')$: $s\_1\_sg^{cf}(X,Y)$:- $X > 10$ & $\texttt{flat}^{cf}(X,Y)$.
$(M5'')$: $s\_2\_sg^{cf}(X,U)$:- $X > 10$ & $\texttt{up}^{cf}(X,U)$.

$M5'$ and $M5''$ are then used in

$(M1')$: $sg^{cf}(X,Y)$:- $s\_1\_sg^{cf}(X,Y)$.
$(M2')$: $sg^{cf}(X,Y)$:- $s\_2\_sg^{cf}(X,U)$ & $sg^{bf}(U,V)$ & $\texttt{down}^{bf}(V,Y)$.

$(M6')$: $m\_sg^{bf}(U)$:- $s\_2\_sg^{cf}(X,U)$.

$\texttt{flat}^{cf}$ and $\texttt{up}^{cf}$ are the grounding subgoals in $M5'$ and $M5''$ respectively. The program M' consisting of rules $\{M1', M2', M3, M4, M5', M5'', M6', M7\}$ is the output of our algorithm. A reader familiar with [BR87] would recognize $s\_1\_sg^{cf}$ and $s\_2\_sg^{cf}$ to be supplementary magic-sets. □

[4] The use of m_sg^{cf} in $M6$ is through $M2$, and we will ignore it.

A second contribution of this paper is to provide a framework for comparing the merits of different classes of adornment patterns. We have already mentioned a *"bcf"* adornment class extending the *"bf"* adornment class. Other adornment classes may be defined to capture restrictions. The adornment class determines the level of detail at which propagated information is described, and thus determines the amount of information available for subsequent compile-time optimizations. Further, while the adornment phase is guided by a SIPS, what happens in practice is that the SIPS for a rule is itself chosen dynamically during the adornment phase. A more descriptive class of adornments can enable a better choice of SIPS.

How do we know whether an adornment pattern is *better* than another? For example, we would like to say that our bcf adornment pattern is *better* than the bf adornment pattern. How do we know that an adornment pattern does indeed capture the restrictions it is intended for? Does the bcf pattern capture all types of conditions? We define *Descriptiveness* of adornments and a *Faithful Adornment Property* to provide an answer to these questions.

We study the faithfulness of the bcf adornment pattern, and examine its strengths and limitations. More refined adornment patterns can be devised to overcome these limitations. The refinements can enable us to use a wider class of restrictions during bottom-up evaluation by suitably refining the Magic rewriting phase.

The rest of the paper is organized as follows. Section 2 defines the magic-sets transformation of [BR87] and some related concepts. The bcf adornment class is introduced in Section 3. We define bcf SIPS, and discuss an algorithm to adorn a program using the bcf SIPS. In Section 4 we present our ground magic-set transformation. Section 5 describes the faithful adornment property. Related work is discussed in Section 6 while Section 7 presents conclusions.

2 Definitions

bf Adornment Class: We define the bf adornment class as one that distinguishes between bound (b) and free (f) argument positions. For the purposes of this paper, bound and free are interpreted as follows: If an argument position in a body literal of a rule is bound, then the variable in it must be bound to some constant in each goal (the constant could be different for different goals) generated from this literal. If the argument position is free, then the variable in that argument position can assume any value in a goal generated from the literal, so it could be free, restricted by a condition, or bound to a constant.

The bf adornment class has been used in [BMSU86, BR87, BKMR89].

SIPS: A Sideways Information Passing Strategy is a decision on how to pass information sideways in the body of a rule while evaluating the rule.

Formally, a SIPS for a rule r with head p^a (a is the head adornment) is defined in [BR87] as a directed labelled graph. The edges of the graph induce a partial order (SIPS order) in which the body literals are to be evaluated, while the labels indicate the bindings to be passed from one literal to another. Bindings are passed only if an equality predicate exists between subterms appearing in the arguments of the two literals. The interested reader is referred to [BR87]. We will refer to the SIPS of [BR87] as bf SIPS, to distinguish them from bcf SIPS defined in Section 3.2, and to highlight the observation that they only consider how *bindings* are passed, ignoring conditions.

A SIPS can be *full*, meaning that all eligible bindings are passed sideways. A full SIPS induces and is defined by a total ordering on the body literals.

Strata: The dependency graph of a Datalog Program is a directed graph whose nodes correspond to the program's predicates. This graph has an edge $q \rightarrow p$ whenever there is a rule in P with p in the head and q in the body. Predicates in an SCC (Strongly Connected Component) of the dependency graph are said to be mutually recursive. Replacing SCC's by a single node yields the reduced dependency graph, which is always a dag. A topological sort on the reduced dependency graph assigns a stratum number to each SCC. By convention, the EDB[5] predicates are placed in stratum 0. We refer the reader to [Ull88] for details.

Magic-sets Transformation: We now define the magic-sets transformation as described in [BR87]. The magic-sets transformation is a two phase transformation.[6] In the first step, we produce an *adorned* program, P^{ad}, in which predicates are adorned with an annotation that indicates which arguments are bound (to constants) and which are free. For each predicate, we have an adorned version that corresponds to all uses of that predicate with a binding pattern that is described by the adornment; different adorned versions are treated as different predicates (and possibly solved differently). For example,

[5] Extensional DataBase, or base predicates.

[6] In [MFPR90] we introduce a one-phase variant, and compare it with the two phase algorithm.

p^{bf} and p^{fb}, are treated as (names of) distinct predicates. An *adornment* for an n-ary predicate is defined to be a string of b's and f's. Argument positions that are treated as free (have no predicate on them) are designated as f, and positions that are bound to a finite set of given values (by equality predicates) are designated as b. The adornment phase is guided by a choice of SIPS for the rules of the program P.

In the second step we transform P^{ad} to produce a magic program M as follows:

1. Initially M is empty.

2. Create a new predicate m_p (the magic predicate for p) for each predicate p in P^{ad}. The arity is the number of bound arguments of p.

3. For each rule in P^{ad}, add a *modified version* to M. If rule r has head, say, $p(\bar{t})$ ($\bar{t}$ is shorthand for all the attributes of p), the modified version is obtained by adding the literal $m\_p(\overline{t^b})$ into the body ($\overline{t^b}$ denotes the arguments of $p(\bar{t})$ that are bound.).

4. For each rule r in P with head, say, $p(\bar{t})$, and for each literal $q_i(\overline{t_i})$ in its body, add a *magic-rule* to M. The head is $m\_q_i(\overline{t_i^b})$. The body contains all literals that precede q_i in the SIPS associated with r, and the literal $m\_p(\overline{t^b})$.

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

EXAMPLE 2.1 Consider the query

$(Q2)$: ?- $sg(\text{john}, Z)$.

on the program P of Example 1.1. Assume the SIPS for rule $P2$ for the goal sg^{bf} requires that the head binding be passed into up, that up pass the binding on U into the sg subgoal, and that sg then pass the binding on V into down.

Rules $A3$ and $A4$ then give the adorned program P^{ad}, with

?- $sg^{bf}(\text{john}, Z)$.

being the adorned query.

Rules $M3$ and $M4$ are the modified rules, $M7$ is the magic rule, and $m\_sg^{bf}(\text{john})$ is the seed fact of the magic program M. □

The careful reader will notice that some joins are repeated in the bodies of rules defining magic predicates and the modified rules (For example, see $M7$ and $M4$). The *supplementary* version of the magic-sets transformation essentially identifies these common sub-expressions and stores them (with some optimizations that allow us to delete some columns from these intermediate, or supplementary, predicates). We refer the reader to [BR87] for details.

3 bcf Adornments

Our objective is to develop an adornment class more descriptive than the bf adornment class in order to propagate arithmetic conditions.

3.1 The bcf Adornment Class

The b and f adornments of the bf adornment class allow us to differentiate between arguments that are bound to (one of) a set of constants, and those that are not. We also want to distinguish arguments that are restricted by an arithmetic condition ("*conditioned*"). We therefore introduce a condition (c) adornment. An argument X is given the c adornment if it is restricted by an arithmetic condition, such as $X > 10$ or $X + Y \leq Z$. We further require that the condition should be *independent*:[7] No free or (other) conditioned variable should appear in the condition. Thus, the condition $X > Y$ does not allow us to adorn either X or Y with c, unless one of them is bound. Similarly, Y and Z must be bound for X to be conditioned by $X + Y > Z$.

The bcf adornment class is defined as the class that uses b, f, and c adornments, with the c adornment interpreted as above, and the b and f adornments interpreted as in the bf adornment class.

The c adornment refines the free adornment of the bf class; some arguments adorned f using the bf class may now be adorned c, as in Query (Qa) of Example 1.1. The following table gives the intuition behind the bcf adornments.

Adornment	Allowed Values	Disallowed Values
b	Finite	Infinite
c	Infinite	Infinite
f	Infinite	Finite

3.2 bcf SIPS

Rules ($A1$) and ($A2$) of the adorned program A in Example 1.1 pass conditions from the head into the

[7] We have developed refined adornment patterns that relax the independence constraint. We do not discuss the refinements here.

first subgoal. Similarly in (Qa), the built-in literal, $X > 10$, passes a condition sideways into sg.

The bf SIPS cannot describe passing of conditions in a rule. A stronger class of SIPS needs to be defined to specify how information about conditions is to be passed in a rule. We refer to the new class as bcf SIPS.

A bcf SIPS is a directed labelled graph, similar to the bf SIPS of [BR87]. In a bf SIPS, each edge into a literal represents bindings coming into the literal, with the label giving the bindings. In a bcf SIPS, the idea is the same, except that the incoming edge represents both bindings and conditions being passed into the literal. The edge thus needs two labels, a bound label (β), and a condition label (χ), giving, respectively, the variables that are bound and conditioned.

For example, in query (Qa) of Example 1.1, the edge into sg has the condition label $\chi = X$ while the bound label is empty. The labels on edges into `flat` in $(A1)$ and `up` in $(A2)$ are similar.

To define bcf SIPS formally, we need to introduce some terminology. Consider the rule:

$$(t): \quad p \text{ :- } q_1 \ \& \ q_2 \ \& \ \ldots \ \& \ q_m \ \& \ c_1 \ \& \ c_2 \ \& \ \ldots \ \& \ c_n.$$

where each q_i is an ordinary[8] literal and each c_j is a built-in literal of the form X op Y, or of the form X op c, where $\mathsf{op} \in \{<, \leq, \geq, >\}$, X and Y are variables, and c is a constant.

Let a be a head adornment for p. Then

p_h^a: is a special literal denoting the head predicate p with the adornment a.

$L(r, a)$: is the set of all literals in rule r along with the special literal p_h^a. For example, $L(t, a) = \{p^a, q_1, q_2, \ldots, q_m, c_1, c_2, \ldots, c_n\}$.

$OL(r)$: is the set of ordinary literals in a rule r. For example, $OL(t) = \{q_1, q_2, \ldots, q_m\}$.

var: For u equal to a rule, a literal, or a set of literals, $\mathsf{var}(u)$ denotes the set of variables that appear in u.

bvar: For u equal to a literal or a set of literals, $\mathsf{bvar}(u)$ denotes the subset of var(u) that can be considered bound after u is solved.

cvar: For u equal to a literal or a set of literals, $\mathsf{cvar}(u)$ denotes the subset of var(u) that can be considered conditioned after u is solved.

EXAMPLE 3.1 For the rule

$$\begin{aligned}(r): \quad p(X, Y) \text{ :- } & X > 10 \ \& \ Z > X \ \& \ S > 10 \ \& \ S < 20 \\ & \& \ q_1(S, X, U, Z) \\ & \& \ V < U \ \& \ q_2(T, V, W, Y, Z) \\ & \& \ W < T.\end{aligned}$$

$OL(r) = \{q_1(S, X, U, Z), q_2(T, V, W, Y, Z)\}$, and $\mathsf{var}(r) = \{S, U, V, W, X, Y, Z\}$.

$\mathsf{bvar}(\{p_h^{fb}, X > 10, Z > X, S > 10, S < 20\}) = \{S, Y\}$. The binding on S can be deduced from the two conditions on S, and Y is bound by the head literal. $\mathsf{bvar}(\{Z > X, V < U, q_1(S, X, U, Z)\}) = \{S, X, U, Z\}$.

$\mathsf{cvar}(\{p_h^{fb}, X > 10, Z > X, S > 10, S < 20\}) = \{X, Z\}$. The condition on X is given, while the condition $Z > 10$ can be deduced. $\mathsf{cvar}(\{X > 10, V < U, W < T, q_1(S, X, U, Z)\}) = \{V\}$. U is bound, so that $V < U$ implies a condition on V. $W < T$ does not condition either W or T. □

bvar gives us the bound variables in a rule as the rule's literals are solved. Clearly in the head p_h^a, the variables in the arguments adorned b are bound. When an ordinary literal is solved, all its variables become bound. Built-in literals do not usually bind variables, unless a deduction system is used.

cvar gives us the conditioned variables of a rule as the rule's literals are solved. When the rule is invoked, only the variables in the condition arguments of the head literal are conditioned. $\mathsf{cvar}(u)$ gives the variables that are conditioned but not bound after solving a set u of literals. An ordinary literal binds all variables that appear in it. A built-in literal containing only one variable generates a condition on that variable. Thus $X > 10$ causes X to be conditioned. A built-in literal with two variables $X > Y$ does not condition either variable independently of the other. However, if Y happens to be bound when $X > Y$ is considered, an independent condition on X is created. Also if Y happens to be conditioned, the condition on Y along with $X > Y$ can sometimes imply an independent condition on X.

We assume that procedures to compute bvar and cvar are available.

Definition 3.1 SIPS: The SIPS for a rule r and a head adornment a, denoted by $\mathsf{sips}(r, a)$, is a directed labelled graph (V, E, B). The vertex set has a node for each literal in $OL(r)$, and a node for each subset of $L(r, a)$. An edge e in E is of the form $T \to q$, where $T \subseteq L(r, a)$ and $q \in OL(r)$, and indicates that information is passed from the literals in T to the ordinary body literal q. The label set B assigns two labels to each edge e: $\beta(e)$ (bound label) and $\chi(e)$ (condition label). Each label is a set of variables. The intuition behind this is that the variables of $\beta(e)$ are bound and the variables of $\chi(e)$ are conditioned by the literals in T, and the SIPS specifies that these bindings and conditions are to be used in evaluating q. B and E must satisfy the following constraints:

[8] A literal g is ordinary if g is not built-in.

1. For each edge e, $\beta(e) \subseteq \mathsf{var}(q)$, $\chi(e) \subseteq \mathsf{var}(q)$, $\beta(e) \subseteq \mathsf{bvar}(T)$, and $\chi(e) \subseteq \mathsf{cvar}(T)$, and $\beta(e) \cap \chi(e) = \emptyset$.
2. E induces a relation on the ordinary literals of the rule r as follows. If $T \to q \in E$, then for every literal $u \in T$, let $u \prec q$ (since information from literals in T is passed to q).

   The relation $\prec$ must be a partial order.

□

$\mathsf{sips}(r, a)$ suggests a computation for a rule r with head adornment a. Condition 2 ensures that the literals of r can be evaluated in some order, and the bindings so obtained can then be passed onto the literals to be evaluated later. The partial order $\prec$ places a constraint on the order of evaluation of literals of r.

3.2.1 Full bcf SIPS

When solving for a subgoal g during evaluation of a rule r, we often want to use all available information. When the information consists of bindings and independent conditions, we want to use all variable bindings obtained by solving previous subgoals, and we want to use all independent conditions on the variables of subgoal g that can be deduced from the built-in subgoals of r in conjunction with the head adornment and the available bindings.

A SIPS that passes all available information is called a full SIPS. We observe that a full SIPS induces a total order on the ordinary subgoals of a rule, and conversely, an ordering of the ordinary subgoals completely specifies a full SIPS. We assume here that a particular head adornment is being considered.

For the rule

(t): p :- q_1 & q_2 & ... & q_m & c_1 & c_2 & ... & c_n.

let $q_1, q_2, \ldots, q_m$ be a given ordering of the ordinary subgoals. We want to determine the earliest position at which a built-in subgoal can be used. If c_i can be used to evaluate subgoal q_j, but none of the previous subgoals, define $\mathsf{eligibility}(c_i)$ to be j. For example, in rule r of Example 3.1, $\mathsf{eligibility}(X > 10) = 1$, $\mathsf{eligibility}(Z > X) = 1$, $\mathsf{eligibility}(V < U) = 2$, and $\mathsf{eligibility}(W < T) = 3$. With only two ordinary subgoals, an eligibility of 3 means that $W < T$ cannot be used to restrict any subgoal.

Let $C(j)$ denote the set of built-in literals with eligibility j, and let $N(j)$ be the set of $c_i' s$, that, when taken in conjunction with $\{p_h^a, C(1), q_1, \ldots, C(j-1), q_{j-1}\}$ generate bindings or conditions for variables of q_j that could not be generated if we were to remove a literal from $N(j)$. $C(j)$ is then the union of $N(j)$ and the set of built-in literals all whose variables appear in $\mathsf{bvar}(\{p_h^a, C(1), q_1, \ldots, C(j-1), q_{j-1}, N(j)\})$.

EXAMPLE 3.2 For example, in rule r of Example 3.1, $C(1) = N(1) = \{X > 10, Z > X, S > 10, S < 20\}$. If r also had the literal $Y > 10$ and the head adornment was p^{fb}, $Y > 10$ would not be in $N(1)$, but it would be included in $C(1)$. $C(2) = N(2) = \{V < U\}$, $N(3) = \emptyset$ and $C(3) = \{W < T\}$. □

Definition 3.2 Full SIPS: A SIPS $\mathsf{sips}(t, a) = (V, E, B)$ is *full* if the following conditions hold: (1) For every literal $q \in OL(r)$, there is exactly one edge of the form $(T \to q) \in E$, (2) The relation $\prec$ induced by E on $OL(r)$ is a total order, (3) If c is a built-in literal of t with eligibility j, then $(\forall k)_{j \le k \le m}(((T \to q_k) \in E) \Rightarrow (c \in T))$, and (4) For every edge $(e = (T \to q) \in E)$, $p_h^a \in T$, $\beta(e) = \mathsf{bvar}(T) \cap \mathsf{var}(q)$, and $\chi(e) = \mathsf{cvar}(T) \cap \mathsf{var}(q)$. □

3.3 Adorning with the bcf Class

3.3.1 Adorning a Rule

Given a SIPS $\mathsf{sips}(r, a) = (V, E, B)$ for a rule r, an adorned version r^{ad} can be computed as follows. For each ordinary subgoal, q_j, we compute two sets, $\mathcal{B}$ and $\mathcal{C}$, using the labels on edges into q_j. $\mathcal{B} = \bigcup_{(e=T \to q_j) \in E}(\beta(e))$ is the set of bound variables, and $\mathcal{C} = (\bigcup_{(e=T \to q_j) \in E}(\chi(e)) - \mathcal{B})$ is the set of conditioned variables passed into q_j. The k^{th} argument position of q_i is given the adornment b if it contains a constant or a bound variable $\in \mathcal{B}$, the adornment c if it contains a conditioned variable $\in \mathcal{C}$, and the adornment f otherwise.

Often, we are given a rule r with a head adornment a, but the SIPS to be used for adorning the rule is not available; an appropriate SIPS has to be selected. The subgoal ordering algorithm of [Mor88] uses a *Bound is Easier* assumption to derive a *full bf sips* for a rule r as the rule is adorned. The subgoal order is determined incrementally, using information on variables bound by the previously ordered subgoals. The selection is done by a next function; this function determines the adornment on each of the remaining subgoals (if the subgoal was to be evaluated next), and selects the next subgoal using a simple cost heuristic (such as "Bound is Easier").

We give an extension of the [Mor88] algorithm for bcf SIPS. The idea is the same, but there is extra work required to track the conditioned variables of the rule. The algorithm depends on a next function. Assuming that the first $j-1$ ordinary subgoals in the SIPS for the rule

(r): p :- q_1 & q_2 & ... & q_m & c_1 & c_2 & ... & c_n.

have been determined to be $u_1, u_2, \ldots, u_{j-1}$, we also have available the set $C = C(1) \cup C(2) \cup \ldots \cup C(j-1)$ of eligible built-in literals, and the sets $\mathcal{B} = \mathsf{bvar}(\{p_h^a, C, u_1, u_2, \ldots, u_{j-1}\})$ and $\mathcal{C} = \mathsf{cvar}(\{p_h^a, C, u_1, u_2, \ldots, u_{j-1}\})$. The function next takes $\mathcal{B}, \mathcal{C}$, and C as input, along with the remaining ordinary and built-in subgoals, and determines the j^{th} subgoal in the SIPS. Presumably next computes, for each choice q for the j^{th} subgoal, the set $C(j)$, uses it to compute $\mathcal{B}$ and $\mathcal{C}$, uses them to determine the adornment on q, and then applies a cost estimate to return the best choice for the j^{th} subgoal.

Algorithm 3.1 *(***ARFS***)*

$\mathcal{B} = \mathsf{bvar}(p_h^a); \mathcal{C} = \mathsf{cvar}(p_h^a); C = \emptyset;$

FOR $j = 1$ **TO** m **BEGIN**

$u_j = \mathsf{next}(\mathcal{B}, \mathcal{C}, C);$

$C(j) = \{c_i \mid \mathsf{eligibility}(c_i) = j\};$

$C = C \cup C(j);$

$\mathcal{B} = \mathsf{bvar}(\{p_h^a, C, u_1, u_2, \ldots, u_{j-1}\});$

$\mathcal{C} = \mathsf{cvar}(\{p_h^a, C, u_1, u_2, \ldots, u_{j-1}\});$

Adorn the k^{th} argument position of u_i with b if the k^{th} argument contains a constant or a variable $\in \mathcal{B}$, with c if the k^{th} argument contains a variable $\in \mathcal{C}$, and f otherwise.

$\mathcal{B} = \mathcal{B} \cup \mathsf{bvar}(\{u_j\});$

$\mathcal{C} = \mathcal{C} - \mathsf{bvar}(\{u_j\});$

END

Order the literals of rule r in the order $u_1, u_2, \ldots, u_m$. Place the built-in literal c_i, where $\mathsf{eligibility}(c_i) = j$, just before u_j. If $\mathsf{eligibility}(c_i) = m + 1$, place c_i after u_m.

$\diamond$

The last step of **ARFS** (Adorn Rule and Find SIPS) sequences the literals in the rule according to the order determined by the selected SIPS, with each built-in literal going at its eligible position.

EXAMPLE 3.3 Consider the behavior of **ARFS** on the rule r of Example 3.1 with the head adornment p^{fb}.

Given that $\mathcal{B} = Y$, next realizes that the choice for the first subgoal is between q_1^{bcfc} and q_2^{fcfbc}. With one binding and two conditions on each, next might choose q_1^{bcfc} as it has fewer free arguments.

With $u_1 = q_1$, **ARFS** computes $C(1) = \{X > 10, Z > X, S > 10, S < 20\}$, $\mathcal{B} = \{Y, S\}$, and $\mathcal{C} = \{X, Z\}$, leading to the adornment q_1^{bcfc}. For the next iteration, $\mathcal{B} = \{Y, S, X, U, Z\}$ and $\mathcal{C} = \emptyset$.

q_2^{fcfbb} is the only choice available to next for the second subgoal. **ARFS** then computes $C(2) = \{V < U\}$, $\mathcal{B} = \{Y, S, X, U, Z\}$, and $\mathcal{C} = \{V\}$, leading to the adornment q_2^{fcfbb}. For the next round, $\mathcal{B} = \{Y, S, X, U, Z, T, V, W\}$ and $\mathcal{C} = \emptyset$.

Since there are no more subgoals, the **FOR** loop terminates. The adorned rule, with subgoals ordered according to the SIPS, is

$$(r^{ad}): \quad p^{fb}(X, Y) :- X > 10 \ \& \ Z > X \ \& \ S > 10 \ \& \ S < 20 \ \& \ q_1^{bcfc}(S, X, U, Z) \ \& \ V < U \ \& \ q_2^{fcfbb}(T, V, W, Y, Z) \ \& \ W < T.$$

$\square$

3.3.2 Adorning a Program

Given a query Q

$$(Q): \quad ?\text{-}\ q(\overline{X}).$$

on a program P, we give an algorithm to create an adorned program P^{ad} for the query goal $q^{\overline{f}}(\overline{X})$.

We maintain a set *int* of interesting adorned goals that need to be solved, and a set *seen* of adorned goals that have already been solved. Initially the query goal $q^{\overline{f}}$ is the only interesting goal.

Algorithm 3.2 (AP)

$int = \{q^{\overline{f}}\}; seen = \emptyset;$

REPEAT

Remove a goal $g = p^a$ from int; $seen = seen \cup \{p^a\}$;

Use **ARFS** to create an adorned version of each rule for p in P, for the adornment p^a;

For each adorned subgoal u^b in a newly adorned rule for p^a, if $(u^b \notin seen) \wedge (u^b \notin int)$, then add u^b to int;

UNTIL $int = \emptyset$;

$\diamond$

Algorithm **AP** (Adorn a Program) is similar to the algorithm in [Mor88] for generating a rule-goal graph.

Two programs $P1$ and $P2$ are said to be query equivalent with respect to a query Q if the query Q produces the same answers when evaluated on $P1$ and $P2$.

Proposition 3.1 *Algorithm* **AP** *terminates on any Datalog program P for any query Q. The adorned program P^{ad} is query equivalent to the original program P with respect to the query Q.* $\square$

4 Ground Magic-sets

Given a program P^{ad} adorned with the bcf adornment class with some choice of SIPS for each rule, we define a transformation that passes information, including conditions, according to the chosen SIPS during a bottom-up evaluation of the rewritten program M. We call our transformation the Ground Magic-sets Transformation, GMT for short.

GMT is similar to the magic-templates transformation of [Ram88]. Given the same generation program A of Example 1.1, magic-templates rewrites A as M. As we saw, M has a rule $(M5)$ that is not range-restricted. GMT rewrites A into the program M' instead, all of whose rules are range-restricted.

A rule is range-restricted if $\mathsf{var}(head) \subseteq \mathsf{bvar}(body)$. In other words, every head variable must be bound by the body literals (usually by appearing in an ordinary subgoal). A program is range-restricted if all of its rules are range-restricted. Range-restrictedness is a sufficient condition for program safety ([Ull88]).

Bottom-up evaluation of programs that are not range-restricted requires us to store non-ground tuples,[9] possibly with conditions on or between variables of the tuple,[10] and to use unification to find satisfying substitutions in a rule. With range-restricted programs, only ground tuples need to be stored, and *matching*[11] can be used instead of unification.

Well-known commercial and experimental relational (System R, DB2, Starburst, Ingres) and deductive (NAIL!, LDL) DBMS do not support non-ground tuples. Application queries are currently mostly range-restricted, and performance is of prime importance. One can expect better performance from ground tuples, not only because matching can replace unification, but also because fast access methods like hashing and indexing are difficult to extend to non-ground tuples. Moreover, an additional problem, subsumption,[12] has to be solved when adding non-ground tuples to a relation. Thus, if a transformation is to be useful in a wide class of databases, it is critical that it preserve the range-restrictedness of a program. GMT satisfies this property.

4.1 An Overview of GMT

To understand GMT, it is best to think of it as a two step transformation. In the first step, we take a range-restricted bcf adorned program (Program A in Example 1.1) and apply the magic-template transformation of [Ram88] to get a program (M) that may have non-range-restricted magic-rules ($M5$). In the second step, we *ground* the magic-rules to get a range-restricted program (M'). We explain the second step through Example 1.1.

The relation m_sg^{cf} defined by rule $M5$, is infinite. However, only a finite number of the values are ever needed during an evaluation, since m_sg^{cf} is only used in $M1$ and $M2$[13], where the magic values are limited by the EDB relations flat and up respectively. We refer to the limiting relations as the *grounding subgoals.* GMT *grounds* the magic-sets rule individually for the two rules $M1$ and $M2$ that use the magic-set. The grounding is done by moving the grounding subgoals (A subgoal g that does not itself limit a conditioned variable, but passes information into a grounding subgoal $\mathcal{G}$, must be moved into the magic-rule along with $\mathcal{G}$; such a subgoal g is also considered to be a grounding subgoal.) into the magic rule. After grounding, rules $M5'$ and $M5''$ are generated for relations $s\_1\_sg^{cf}$ and $s\_2\_sg^{cf}$, that can be treated as supplementary magic-sets for rules $M1$ and $M2$. $M1$ and $M2$ are rewritten to use the supplementary magic-sets rather than the magic-sets, generating $M1'$ and $M2'$. The magic-rules for the non-grounding subgoals ($M6$) are now rewritten to use the supplementary magic-sets ($M6'$). Though it is not apparent in Example 1.1, magic-rules for any grounding subgoals must also be rewritten.

Section 4.3 presents the GMT algorithm. This algorithm implements the two step approach, and avoids rewriting magic-rules by generating them in a particular order.

4.2 GMT Groundability

In this section we state the conditions under which an adorned program can be transformed by the GMT algorithm through grounding of the magic-template rules. We also indicate how such an adorned program may be obtained from a given program and query.

4.2.1 Allowable Grounding Subgoals

EXAMPLE 4.1 Consider the query:

(Q): ?- $u(X)$ & $Y > X$ & $p^{bc}(X,Y)$.

$(r1)$: $p^{bc}(X,Y)$:- $q^{bf}(X,Z)$ & $p^{bc}(Z,Y)$.
$(r2)$: $p^{bc}(X,Y)$:- $v(X,Y)$.

p^{bc} is a grounding subgoal in $r1$ as it limits the conditioned variable, Y. q^{bf} is also grounding, since it passes a binding into p^{bc}. The magic-rule from the use of p^{bc} in Q

(m): $m\_p^{bc}(X,Y)$:- $u(X)$ & $Y > X$.

will be grounded to

(sm): $s\_1\_p^{bc}(X,Y)$:- $u(X)$ & $Y > X$ & $q^{bf}(X,Z)$ & $p^{bc}(Z,Y)$.

Rule sm generates new magic values for p^{bc} with the magic-rule:

$(m1)$: $m\_p^{bc}(Z,Y)$:- $u(X)$ & $Y > X$ & $q^{bf}(X,Z)$.

[9] A tuple is non-ground if it contains variables. For example, the tuple $p(X,5)$ is non-ground. A tuple is ground if it does not have a variable.

[10] such as the tuple $p(X,5)$ & $X > 5$.

[11] When one of the two terms to be unified does not have variables, the task is easier, and is referred to as *matching*.

[12] Is a tuple subsumed in another?

[13] The use in $M6$ is through $M2$, and we ignore it temporarily.

that will be grounded to

$$(sm1): \quad s\_1\_p^{bc}(Z,Y) :- u(X) \ \& \ Y > X \ \& \ q^{bf}(X,Z) \ \& \ q^{bf}(Z,Z1) \ \& \ p^{bc}(Z1,Y).$$

More magic values for p^{bc} are generated, and the grounding process feeds into itself. GMT will not terminate. □

To avoid the termination problem of Example 4.1, we require that a grounding subgoal in a rule for p^a be non-recursive with p^a. Thus in Example 4.1, p^{bc} cannot be used as a grounding subgoal in $r1$. $r1$ will then have no grounding subgoals, and we will treat the condition on Y as unusable.

GMT can be extended to allow grounding by recursive subgoals in Datalog programs. In Example 4.1, we could compute the magic values for the first argument separately, and then combine it with the condition that is invariant across recursion. We do not discuss the extension in this paper. However, in the presence of function symbols, there are examples where recursive grounding subgoals must be avoided if a grounding magic transformation is to terminate.

4.2.2 Groundable SIPS

In a rule r for a conditioned literal, p^a, some of the subgoals are grounding, and others are non-grounding. The grounding subgoals, required to be non-recursive with p^a, are moved out into the (supplementary) magic-rules by GMT. Hence arbitrary flow of information between the grounding and non-grounding subgoals is not possible. We define *Groundable SIPS* as the SIPS whose information flow requirements can be implemented by GMT.

Definition 4.1 Groundable SIPS: Let r be adorned according to a SIPS $s = \mathsf{sips}(r,a) = (V,E,B)$, where $a = \star c \star$.

SIPS s is said to be *groundable* if the subgoals of r can be partitioned into two sets, $G(s)$ (grounding subgoals) and $NG(s)$ (non-grounding subgoals) satisfying the following properties:

1. p^a and $G(s)$ are non-recursive.
2. No information is passed from $NG(s)$ to $G(s)$. Thus, $(((T \rightarrow q) \in E) \wedge (q \in G(s))) \Rightarrow ((T \cap NG(s)) = \emptyset)$. This condition allows us to move the grounding subgoals out of the rule and solve them separately before solving non-grounding subgoals.
3. If either p_h^a or an element of $G(s)$ passes information into an element q of $NG(s)$, then p_h^a and all of G passes information into q. Thus, $(((T \rightarrow q) \in E) \wedge (q \in NG(s)) \wedge ((T \cap (G(s) \cup p_h^\alpha)) \neq \emptyset)) \Rightarrow (T \supseteq (G(s) \cup p_h^\alpha))$.
4. The head literal p_h^a does not pass any condition into a literal of $NG(s)$. Consequently, head conditions can be used only in the literals of $G(s)$.
5. $\mathsf{var}(G(s)) = \mathsf{bvar}(G(s))$. Thus, the variables in a built-in literal of $G(s)$ must be bound by $G(s)$.

For a groundable SIPS s,
$\mathsf{unusable}(s) = (\mathsf{cvar}(p_h^a) - \mathsf{bvar}(G(s)))$.
□

The conditioned variables in the head that are not referenced in the grounding set $G(s)$ are considered to be *unusable* in restricting computation of the rule. A groundable SIPS does not pass the unusable conditions into the $NG(s)$ subgoals, and thus may not be a full SIPS. Conditions on unusable variables cannot be grounded in the magic rule. GMT just drops the unusable conditions in the grounding phase.

EXAMPLE 4.2 The SIPS in Example 1.1 are groundable. $\{flat^{cf}\}$ is the grounding set in rule $A1$. $\{up^{cf}\}$ is the grounding set in rule $A2$. The unusable sets are empty.

The SIPS $s1$ for $r1$ in Example 4.1 is not groundable. The subgoal p^{bc} can only be in the $NG(s)$ partition due to Constraint 1, and the head condition is passed into p^{bc}, violating Constraint 4. An alternative SIPS $s2$ that is similar to $s1$ except that it does not pass the head condition into p^{bc}, is groundable with $G(s) = \emptyset$, and $\mathsf{unusable}(s2) = \{Y\}$. Note that $s2$ is not a full SIPS. □

Theorem 4.1 *Given a conditioned goal p^a, let r be a rule for p^a adorned with a groundable SIPS s, and let mr be the non-range-restricted rule for m_p^a obtained by the magic-templates transformation. Remove from m_p^a the arguments corresponding to* $\mathsf{unusable}(s)$, *and ground mr with the literals $G(s)$. The resulting rule sr is range-restricted.* □

The importance of this theorem is that once the grounding subgoals $G(s)$ of a rule r for p^a are determined, the same set $G(s)$ can be used to ground all magic-rules for p^a. We do not need a different set of grounding subgoals for each magic-rule.

4.2.3 Groundable Programs

In Section 3.3 we gave an algorithm to adorn a program P, choosing a full SIPS for each rule as we adorned the rule. We show that the algorithm can be modified to chose a groundable SIPS for each rule. A program adorned according to groundable SIPS is said to be groundable.

Proposition 4.1 *There exists a groundable SIPS for every rule r.* □

Proof: Given a rule r a groundable sips s with $G(s) = \emptyset$ and $\mathsf{unusable}(s) = \mathsf{cvar}(p_h^a)$ can easily be designed. □

However, a SIPS s with $\mathsf{unusable}(s) = \mathsf{cvar}(p^a_h)$ is usually not interesting. We prefer a SIPS that (i) has few $\mathsf{unusable}$ variables (so that more head conditions are used), and (ii) has a small grounding set (so that fewer literals are copied into each magic-rule).

The **ARFS** algorithm of Section 3.3.2 will produce groundable SIPS if we bias the next function towards choosing non-recursive literals, and if we do not use head conditions after a recursive literal has been chosen.

4.3 GMT Algorithm

In this subsection we describe the Ground Magic Transformation. Section 4.1 gave an overview of the algorithm on the Same Generation example, and the reader should be able to apply the description to that example. We introduce notation, give the algorithm, and then work out an example.

During the GMT transformation, we generate magic-rules (for magic-predicates m_p^a) that are not range-restricted. The non-range-restricted magic-rules are later grounded, once for each rule r for p^a. The grounded rules are called supplementary-rules, and the magic-predicates extended with relevant arguments of grounding subgoals from rule r are called supplementary-predicates, $s\_r\_p^a$. By Theorem 4.1, supplementary-rules are range-restricted. The adorned rule r is modified by replacing the grounding subgoals with the supplementary predicate $s\_r\_p^a$.

The SIPS for the newly constructed supplementary-rules for $s\_r\_p^a$ as well as for the new transformed rule r' for p^a can be derived from the groundable SIPS for r. For a grounding subgoal, any information that came from the head earlier, will now come from the body of the magic-rule into which the grounding subgoal is placed. For a non-grounding subgoal, any information that came from the head or a grounding subgoal will now come from the supplementary predicate. The actual translations of the SIPS are straightforward, and are left to the reader.

Each of the grounding (G) and non-grounding (NG) subgoals of rule r for p^a have magic-rules due to their appearance in r. In a magic transformation without grounding, these magic-rules use the subgoal m_p^a. However, m_p^a does not exist after grounding; we only have the grounded supplementary predicates. The magic-rules for G and NG therefore need to be rewritten.

In GMT, we generate the magic-rules in a particular order that avoids the need to rewrite them later. We use the following idea:

Subgoals in a rule for p belong to lower or same SCC's as the SCC for p. As a result, a magic-set transformation on a rule for p cannot generate magic-rules for a predicate of a higher SCC. Thus, we can process a program P from the top SCC (query goal) to the lowest SCC (EDB's), confident that after we process the SCC for a predicate p, grounding in lower SCC's will not effect the magic-rules for p.

We introduce some notation.

Given a program P and a query Q, let k be the stratum of the SCC of the query predicate. Let $P_1, P_2, \ldots, P_k$ be the rules in P for predicates of stratum $1, 2, \ldots k$, $\mathsf{preds}(j)$ be the predicates in stratum j, $\mathsf{stratum}(p^a)$ be the stratum of predicate p^a, and let $P(p^a)$ (or $P_j(p^a)$) be the rules for predicate p^a in the program P.

Magic-rules for the magic-predicate m_p^a of a predicate p^a of stratum j are generated from usage of p^a in strata higher than or equal to j. We denote the magic-rules generated from higher strata by $M_h(p^a)$. Magic-rules generated from the same stratum ($= j$) are denoted by $M_=(p^a)$.

Let $M(p^a) = M_h(p^a) \cup M_=(p^a)$.

A rule mr in $M(p^a)$ will not be range-restricted if a includes a c adornment. Each conditioned variable X of p^a_h will be unlimited (not bound), appearing in the rule body in a built-in literal of the form X op c, where c is a constant, or X op Y, where Y is a limited (bound) variable.

Let $\mathcal{M}(j) = \bigcup_{p \in \mathsf{preds}(j)} M(p)$ be the set of magic-rules for all predicates of stratum j. Similarly, $\mathcal{M}_h(j)$ and $\mathcal{M}_=(j)$ are the unions of $M_h(p)$ and $M_=(p)$ over predicates in stratum j.

We use m_p^a for the magic-predicate of p^a, and $s\_r\_p^a$ for the supplementary-predicate of rule r for p^a.

Algorithm 4.1 (GMT)

INPUT:

- A range-restricted, groundable program P, bcf adorned for the query $Q = c \,\&\, q^\beta(\overline{X})$, where c denotes a set of built-in literals.
- For each rule r for an adorned predicate p^a in P, a groundable SIPS $s = \mathsf{sips}(r, a)$.
- A stratification of Program P, with query predicate q in stratum k, and the EDB's in stratum 0.

OUTPUT: A range-restricted magic transformed program $M^g(P) = \mathrm{GMT}(P, Q)$, that is equivalent to P with respect to the query Q.

METHOD:

Create the seed magic-rule

$(M_h(q))$: $m\_q^{\beta}(\overline{X}^{bc})$:- c.

where $\overline{X}^{bc}$ denotes the bound and conditioned arguments of $\overline{X}$.

FOR stratum $j = k$ TO 1, in that order, DO:

(A): Form supplementary predicates.
For every rule $r \in P_j$ of a conditioned predicate $p^a \in \mathsf{preds}(j)$, do:

1. Create a supplementary predicate $s\_r\_p^a$.
2. Determine the arguments, $\overline{Y}$, of $s\_r\_p^a$.
$\overline{Y} = (\mathsf{bvar}(p^a_h) \cup \mathsf{bvar}(G(s))) \bigcap (\mathsf{var}(p^a_h) \cup \mathsf{var}(NG(s)))$.
3. Remove the grounding subgoals G from r, and place the subgoal $s\_r\_p^a(\overline{Y})$ in r instead.

(B): Magic-Transform P_j.
Do the magic-template transformation on every rule $r \in P_j$,[14], thereby creating magic-rules for the magic-predicates of subgoals in r.

The magic-rules generated in this step include rules for magic-predicates of predicates in stratum j. These go into $\mathcal{M}_=(j)$, completing construction of the set $\mathcal{M}(j)$. Magic-rules for magic predicates of predicates in strata $i < j$ are also created in this step. These go into $\mathcal{M}_h(i), i < j$.

(C): Create Supplementary Rules.
For each supplementary predicate $s\_r\_p^a$ created in Step (A), construct the supplementary rules defining $s\_r\_p^a$. A supplementary rule is generated from each magic-rule m for m_p^a, as follows:

1. Unify the head of m with the b and c adorned arguments in the head of rule r. Do the substitution implied by the unification into the body literals of m. Let B be the resulting subgoals in the body of m.
2. Create the supplementary rule:

   (sm): $s\_r\_p^a(\overline{Y})$:- B & $G(s)$.

   where $s = \mathsf{sips}(r, a)$.
3. From amongst the B subgoals of the rule created above, remove any conditions over variables in $\mathsf{unusable(s)}$.

[14]Note that, if r has a condition adornment, r would have been modified in step (A), with the grounding subgoals of r replaced by the supplementary predicate.

(D): Magic-Transform Supplementary-Rules.
Do the magic-template transformation on every supplementary rule created in Step (C), thereby creating magic-rules for the magic-predicates of each grounding subgoal in the supplementary-rule.

Since the grounding subgoals, $G(s)$, in a rule for $s\_r\_p^a$ are required to be in a lower stratum than p^a, all magic-rules generated in this step are for magic-predicates of predicates in strata $i < j$.

The supplementary rules, along with the rules of program P after modification in Step (A3), form the output program M^g. $\diamond$

EXAMPLE 4.3 (GMT): Consider the adorned program P

$(P1)$: $p^{cf}(X,Y)$:- $U > 10$ & $q^{ccf}(X,U,V)$ & $W > V$ & $p^{cf}(W,Y)$.
$(P2)$: $p^{cf}(X,Y)$:- $u^{cf}(X,Y)$.

$(P3)$: $q^{ccf}(X,Y,Z)$:- $q1^{cf}(X,U)$ & $q2^{fc}(W,Y)$ & $q3^{bbf}(U,W,Z)$.

for the query

(Q): ?- $X > 10$ & $p^{cf}(X,Y)$.

The query goal, p^{cf}, is in stratum $k = 2$, q^{ccf} is in stratum 1, and the remaining predicates are EDB's in stratum 0.

Let the SIPS $s1$, $s2$ and $s3$ for the three rules be the full SIPS corresponding to the subgoal order shown, with $G(s1) = \{U > 10, q^{ccf}(X,U,V)\}$, $G(s2) = \{u^{cf}(X,Y)\}$, and $G(s3) = \{q1^{cf}(X,U), q2^{fc}(W,Y)\}$. The $\mathsf{unusable}$ sets are empty.

The initialization step of GMT creates the magic-rule.

$(Mp1)$: $m\_p^{cf}(X)$:- $X > 10$.

GMT now performs two iterations, first for stratum 2 and then for stratum 1. In the following discussion, a paragraph labelled **(jL)** describes the effect of Step **(L)** on stratum **j**.

(2A): $s\_1\_p^{cf}(X,V)$ and $s\_2\_p^{cf}(X,Y)$ are the supplementary predicates for rules $P1$ and $P2$. $P1$ and $P2$ are rewritten as:

$(P1')$: $p^{cf}(X,Y)$:- $s\_1\_p^{cf}(X,V)$ & $W > V$ & $p^{cf}(W,Y)$.
$(P2')$: $p^{cf}(X,Y)$:- $s\_2\_p^{cf}(X,Y)$.

(2B): Magic transformation on $P1'$ yields

$(Mp2)$: $m\_p^{cf}(W)$:- $s\_1\_p^{cf}(X,V)$ & $W > V$.

(2C): We create supplementary-rules for the predicates $s\_1\_p^{cf}$ and $s\_2\_p^{cf}$. Each predicate has two rules, one each from the magic rules $Mp1$ and $Mp2$.

$$(SM1a): \quad s\_1\_p^{cf}(X,V) :- X > 10 \ \& \ U > 10 \ \& \ q^{ccf}(X,U,V).$$
$$(SM1b): \quad s\_1\_p^{cf}(X,V) :- s\_1\_p^{cf}(X_1,V_1) \ \& \ X > V_1 \ \& \ U > 10 \ \& \ q^{ccf}(X,U,V).$$

$$(SM2a): \quad s\_2\_p^{cf}(X,Y) :- X > 10 \ \& \ u^{cf}(X,Y).$$
$$(SM2b): \quad s\_2\_p^{cf}(X,Y) :- s\_1\_p^{cf}(X_1,V_1) \ \& \ X > V_1 \ \& \ u^{cf}(X,Y).$$

(2D): Magic transformation on $SM1a$ and $SM1b$ generates two magic-rules for m_q^{ccf}. We omit magic-rules for EDB predicates from this example.

$$(Mq1): \quad m\_q^{ccf}(X,U) :- X > 10 \ \& \ U > 10.$$
$$(Mq2): \quad m\_q^{ccf}(X,U) :- s\_1\_p^{cf}(X_1,V_1) \ \& \ X > V_1 \ \& \ U > 10.$$

(1A): We form the supplementary predicate $s\_3\_q^{ccf}(X,U,W,Y)$ for rule $P3$, and rewrite $P3$ as

$$(P3'): \quad q^{ccf}(X,Y,Z) :- s\_3\_q^{ccf}(X,U,W,Y) \ \& \ q3^{bbf}(U,W,Z).$$

(1C): Two supplementary-rules for $s\_3\_q^{ccf}$ are generated, one each from the two magic-rules for m_q^{ccf}.

$$(SM3a): \quad s\_3\_q^{ccf}(X,U,V,W) :- X > 10 \ \& \ Y > 10 \ \& \ q1^{cf}(X,U) \ \& \ q2^{fc}(W,Y).$$
$$(SM3b): \quad s\_3\_q^{ccf}(X,U,V,W) :- s\_1\_p^{cf}(X_1,V_1) \ \& \ X > V_1 \ \& \ Y > 10 \ \& \ q1^{cf}(X,U) \ \& \ q2^{fc}(W,Y).$$

GMT terminates, with the six supplementary-rules together with $P1'$, $P2'$ and $P3'$ forming the magic transformed program M^g. □

Theorem 4.2 *Given a range-restricted groundable program P, bcf adorned for the query Q, the program $M^g(P) = \mathrm{GMT}(P,Q)$ has the following properties:*

- *$M^g(P)$ is range-restricted.*
- *$M^g(P)$ is query equivalent to P with respect to the query Q.*
- *A Bottom-Up Evaluation of $M^g(P)$ restricts computation of each predicate according to the groundable SIPS of P.*

□

5 The Faithful Adornment Property

We have seen that more *"descriptive"* adornments enable us to propagate bindings more effectively. In particular, the bcf adornment pattern allows us to propagate arithmetic conditions in certain situations where the bf adornment pattern would not. However, the bcf pattern may not be sufficiently expressive for certain other classes of restrictions. For example, it does not allow us to describe constraints between two or more arguments. The conclusion that we draw is that it is worthwhile to consider several classes of adornment patterns, depending on the information that we wish to propagate. In this section, we examine classes of adornment patterns as objects of interest in their own right, and attempt to identify significant properties of such classes.

In comparing the merits of different classes of adornment patterns and corresponding algorithms for generating adorned programs based on these patterns, the following criteria are important: (i) the class of restrictions (or bindings) that can be described by a given class of adornment patterns, and (ii) the degree of accuracy with which the adorned program predicts the restrictions in goals that are generated at run-time.

The meaning of an adorned literal, say $p^a(t)$ with a in some class A of adornment patterns, can be formalized as an abstraction of the set of invocations of this literal during execution. For example, $p^{bf}(X,Y)$ indicates that the first argument is bound to some constant and nothing is known about the second argument.[15] $p^{bf}(X,Y)$ thus denotes the set of goals in which the first argument is a constant in the domain, while the second argument could be anything. In defining a new class of adornment patterns, such as bcf, we must specify the set of run-time goals that are described by an adorned literal for every adornment of the new class. The finer the resolution of the set of goals, the more *descriptive* the class of adornment patterns will be. Thus, $p^{bc}(X,Y)$ describes the set of p goals in which the first argument is bound to a constant, and the second argument is restricted by an arithmetic condition involving no other argument. We cannot distinguish this set using bf adornments; we are compelled to approximate it by $p^{bf}(X,Y)$, thereby losing the restriction on the second argument.

[15] This is our interpretation of f. In some contexts, such as testing whether unification can be replaced by matching, it may be of interest to determine whether Y is truly a free variable, and then we would have to choose an adornment pattern that lets us state this.

Without loss of generality, we will assume in the rest of this section that all arguments of a literal are distinct variables, with equality being indicated explicitly through condition literals. Let us denote the set of run-time goals described by an adorned literal l under some class of adornments A as $conc(l, A)$, to be read as "the concretization of l under A". When comparing two different adorned literals, we will require that their predicate names be identical; beyond this, the predicate names are not relevant. We will therefore talk freely of the "set of goals described by an adornment pattern" — it should be understood that we mean the "set of goals described by an adorned literal with the given adornment pattern". As an example, $conc(bffcf, bcf)$ is an abbreviation for $conc(p^{bffcf}, bcf)$.

To examine how effectively the information in the head adornment of a rule (say for p^a) can be utilized, we must recognize first that some sort of "worst-case" assumption must be made about a goal, say g, that could invoke the rule. We know that $g \in conc(p^a, A)$. Thus, if the head is $p^{bff}(X, Y, Z)$, we know that $X = d$, for some constant d. However, with the bf adornment pattern, this is all that we can safely assume — it is possible that the condition $Y > 5$ also holds (this goal is also in $conc(p^{bff}, bf)$), but we cannot make use of this binding (since the goal with $X = d$ and Y free is also in $conc(p^{bff}, bf)$ and we cannot distinguish between them using the bff adornment). Let us define the *canonical set of goals for an adorned literal l under an adornment class A*, written as $canon(l, A)$ to be the subset, not necessarily proper, of $conc(l, A)$ such that each goal in $canon(l, A)$ is minimally restricted subject to membership in $conc(l, A)$.

EXAMPLE 5.1 $canon(p^{bff}, bf)$ is the set of goals $p(X, Y, Z)$ where $X = d$ (for any constant d), while Y and Z are free variables. The goal $X = d, Y > 10, Z$ = free variable is in $conc(bff, bf)$, but is not included in $canon(bff, bf)$.

$conc(p^c, bcf)$ is the set of goals $p(X)$ where X is bound to a constant, or X has some condition on it. It could be any condition, weak or strong. $canon(p^c, bcf)$ is the set obtained by excluding from $conc(c, bcf)$ the goals where X is bound. Every goal with some condition on X is in $canon(c, bcf)$. $p(X)$ & $X > 0, p(X)$ & $X > -1000, p(X)$ & $X > -10^{100}$ are all in $canon(c, bcf)$. Note that we do not take a union of all the conditions to get the goal X = free in $canon(c, bcf)$.

$conc(p^{cc}, bcf)$ includes goals $p(X, Y)$ with independent conditions on X and Y. However, the goal $p(X, Y)$ & $X > Y$ is not included. □

For simplicity, we will require that *conc* and *canon* both be specified for all adornment patterns in an adornment class A, as a part of the definition of A.[16]

The *arity* of an adornment pattern l is the number of arguments of a literal adorned by l. Thus $arity(bff = 3), arity(c) = 1, arity(cc) = 2$.

5.1 Descriptiveness

Definition 5.1 Descriptiveness of Adornments: Given an adornment pattern l_1 in adornment class A_1 and an adornment pattern l_2 in adornment class A_2

1. If $conc(l_1, A_1) \subseteq conc(l_2, A_2)$, and every goal in $canon(l_1, A_1)$ is subsumed by a goal in $canon(l_2, A_2)$, define (l_1, A_1) to be **equally or more descriptive** ($\succeq$) than (l_2, A_2), and define (l_2, A_2) to be **equally or less descriptive** ($\preceq$) than (l_1, A_1).
2. If $(l_1, A_1) \succeq (l_2, A_2)$ and $(l_2, A_2) \succeq (l_1, A_1)$, define (l_1, A_1) to be **equally descriptive** ($=$) to (l_2, A_2).
3. In absence of any of the above relations, (l_1, A_1) and (l_2, A_2) are incomparable. In particular, this is so if l_1 and l_2 are literals of different arity.

□

We say that (l_1, A_1) is **more descriptive** ($\succ$) than (l_2, A_2) if $(l_1, A_1) \succeq (l_2, A_2)$ and $(l_1, A_1) \neq (l_2, A_2)$. $\prec$ is defined similarly.

Thus $(b, bcf) = (b, bf)$, $(f, bcf) = (f, bf)$, $(b, bcf) \succ (c, bcf) \succ (f, bf)$, and (f, bf) is incomparable to (ff, bf).

Definition 5.2 Lattice Class: An adornment class A forms a lattice and is said to be a lattice class if it has (1) for every arity n a top adornment $L(n)$ that is more descriptive than any other adornment of arity n in A, (2) for every arity n a bottom adornment $l(n)$ that is less descriptive than any other adornment of arity n in A, (3) a $glb(l_1, l_2)$ operator that gives the most descriptive adornment $l_3 \in A$ such that $l_3 \preceq l_1 \wedge l_3 \preceq l_2$, provided $arity(l_1) = arity(l_2)$. (4) a $lub(l_1, l_2)$ operator that gives the least descriptive adornment $l_3 \in A$ such that $l_3 \succeq l_1 \wedge l_3 \succeq l_2$, provided $arity(l_1) = arity(l_2)$. □

The bf and bcf adornment classes are lattice classes. For each, b is the most descriptive and f the least descriptive adornment of arity 1. In this section, we only consider lattice classes.

Definition 5.3 Descriptiveness of Adornment Classes: An adornment class A_1 is *more descriptive* than A_2 if the following hold: (1) for every adornment pattern $l_2 \in A_2$, there is an equally descriptive pattern $l_1 \in A_1$, and (2) there is some adornment pattern $l_1 \in A_1$ such that no $l_2 \in A_2$ is equally descriptive.

[16] For the bf and bcf adornments, we need only specify *conc* and *canon* for the single letter adornments, since the rest can then be derived. Other adornment classes may not have this property.

Adornment classes A_1 and A_2 are *equally descriptive* if the following holds: (1) For every adornment pattern $l_2 \in A_2$, there is an equally descriptive pattern $l_1 \in A_1$, and (2) For every adornment pattern $l_1 \in A_1$, there is an equally descriptive pattern $l_2 \in A_2$. □

The class of bcf adornment patterns is thus more descriptive than the class of bf adornment patterns (For the pattern $c \in bcf$, there is no equally descriptive pattern in the bf class).

Let us denote the adornments used in [Ram88] by B and f. B simply indicates the possibility of an argument being bound, while f has the usual interpretation. Call this the Bf class of adornments. Since a "B" argument may be free in the worst case, B and f of Bf are equally descriptive. Consequently, the bf class is more descriptive than the Bf class.

While it is clearly desirable that an adornment class be more descriptive, this may greatly increase the size of the adorned program in the worst case.

5.2 Faithfulness

Another important property of adornment classes is how accurately restrictions are *predicted* for run time goals. Indeed, this also depends on the class of SIPS and the adornment algorithm. To motivate the definitions that follow, we consider two examples.

EXAMPLE 5.2 Consider a bCf adornment class. b and f are interpreted as before, but a C adornment on an argument means that the condition may be independent, or it may depend on another argument having the C adornment. $canon(p^{CC}, bCf)$ thus includes the goal $g1 = p(X,Y)$ & $X > 10$ & $Y > 10$ as well as the goal $g2 = p(X,Y)$ & $X > Y$. Note that the bcf class would have disallowed the latter goal. Consider the rule

$(T1)$: $p^{CC}(X,Y)$:- $q1^f(X)$ & $q2^C(Y)$.

First note that it would be incorrect to use the adornment $q1^C$, as the goal $g2$ would invoke the subgoal $q1(X)$ without any condition on X, and such a goal is not in $conc(q1^C, bCf)$.

However, the goal $g1$ invokes the subgoal $q1(X)$ & $X > 10$. This is in $conc(q1^f, bCf)$, but it is also in $conc(q1^C, bCf)$, and $q1^C$ is more descriptive than $q1^f$. The given adornment in rule $(T1)$ is thus not the most descriptive one that can be used for $g1$.

The bCf class has been unable to *accurately* predict the restrictions on run time goals, for some of the least restrictive head goals.

As an aside, note that $(b, bCf) \succ (c, bcf) \succ (C, bCf) \succ (f, bcf)$. The classes bcf and bCf are not equally descriptive, and neither is more descriptive than the other. □

EXAMPLE 5.3 For the rule

$(T2)$: $p^f(X)$:- $q^f(X)$.

adorned using the bf class, let us see what the bcf class can do. Some goals in $conc(f, bf)$ (say, $p(X)$ & $X > 10$), could be better represented by $p^c \in bcf$, without being representable by $p^b \in bf$. Further, for every such goal, the subgoal q would be best described by q^c, which is more descriptive than the q^f.

Thus the rule $(T2)$ can be improved upon by using the bcf class, and the improvements carry over into the subgoals. On the other hand, if rule $(T2)$ were adorned using the bcf class, we couldn't improve it using the bf class.

□

Definition 5.4 Legal Adornments: Given an adornment class A, let the rule

$p^a(t)$:- $q_1^{a_1}(t_1)$ & $\ldots q_n^{a_n}(t_n)$.

be solved with a goal in $conc(a, A)$. If for each i, the subgoal generated for q_i is in $conc(a_i, A)$, then the rule is said to be legally adorned with respect to class A.

A program is legally adorned if every rule in it is legally adorned. An adornment algorithm that produces legally adorned programs is a legal adornment algorithm. □

For example, the rule

$p^f(X)$:- $q^b(X)$.

is not legally adorned.

Definition 5.5 Faithful Rule Adornments: An adorned rule r

(r): $p^a(t)$:- $q_1^{a_1}(t_1)$ & $\ldots q_n^{a_n}(t_n)$.

legally adorned in adornment class A using a SIPS s is said to be faithfully adorned with respect to an adornment class B if condition **(D)** holds for any choice of an adornment $p^b \in B$ that satisfies the two conditions:

1. $(p^b, B) \succeq (p^a, A)$, and
2. if there exists an adornment $p^c \in A$ that is more descriptive than $p^a \in A$, then it is not the case that $(p^b, B) \succeq (p^c, A)$.

(D): Let the rule (r) be solved for any choice of a goal $\mathcal{G}$ in $canon(p^b, B)$ according to the chosen SIPS s, and let $\mathcal{G}_i$ be the goal generated from the i^{th} body literal. Let $q_i^{b_i}$ be the most descriptive adornment in B such that $\mathcal{G}_i \in conc(b_i, B)$. Then, for all i, $(q_i^{a_i}, A) \succeq (q_i^{b_i}, B)$. □

The definition is based on the following intuition: Suppose we find an adornment $b \in B$ that is *sandwiched* between a and another adornment c of A, and the use of this b in the head lets us describe a subgoal by an adornment more descriptive than the one used in r. We then conclude that r could be adorned better using class B. Hence, we say that r is not faithfully adorned with respect to class B.

If $B = A$, the only choice for b is the adornment a.

It is desirable that a rule adorned using a class A is faithfully adorned with respect to A (We shorten this to "(r) is faithfully adorned").

Rule $(T1)$ in Example 5.2 is not faithfully adorned (with respect to class bCf). This formalizes our notion that the bCf class did not accurately predict the restrictions on subgoals of $(T1)$.

Rule $(T2)$ in Example 5.3 is not faithfully adorned with respect to class bcf. If we assume $(T2)$ is adorned using the bcf class, then $(T2)$ is faithfully adorned with respect to class bf.

Definition 5.6 Faithful Adornment Property: Let A and B be classes of adornment patterns, S a class of SIPS over A, and L a legal adornment algorithm. We say that (A, S, L) has the *faithful adornment property with respect to B* if, for every legally adorned program P^{ad} produced by L using some SIPS $s \in S$, every rule in P^{ad} is faithfully adorned with respect to B.

We say that adornment class A has the faithful adornment property with respect to adornment class B if we can define a class of SIPS S and a legal adornment algorithm L such that (A, S, L) has the faithful adornment property with respect to B.
□

A desirable property of an adornment class A is that it have the faithful adornment property with respect to itself (We often shorten this to "A has the faithful adornment property"), for then A will accurately predict the run time restrictions on goals.

The bCf class does not have the faithful adornment property.[17] Example 5.2 gave a rule $(T2)$ that wasn't faithfully adorned.

Proposition 5.1 *The bf adornment class has the faithful adornment property.* □

It is worth noting that the Bf class has the faithful adornment property trivially.

If an adornment class A does not have the faithful adornment property with respect to a different class B, then using B we can pass some restrictions that cannot be passed using A. On the other hand, if B has the faithful adornment property with respect to A, then B is superior to A in passing restrictions.

Proposition 5.2 *The bf adornment class does not have the faithful adornment property with respect to the bcf class.* □

[17] However, the bCf class can be refined into a faithful class that can pass both dependent and independent conditions. Discussion of this is beyond the scope of this paper.

Also, the Bf class is not faithful with respect to the bf class. Both of these results are consequences of the following theorem.

Theorem 5.1 *Let A and B be lattice classes whose bottom adornments are equally descriptive, and let B be more descriptive than A. Then A does not have the faithful adornment property with respect to B.* □

Proof:

We consider the legally adorned rule

$$(r): \quad p^a(X) :- q^a(X).$$

and find an adornment $p^b \in B$ that is more descriptive than $p^a \in A$ and satisfies the condition (2) of Definition 5.5. Then for a goal $g \in canon(p^b, B)$, r generates a subgoal capturable by $q^b \in B$ (or perhaps an adornment even more descriptive than q^b). $q^a \in A$ is less descriptive than $q^b \in B$; hence the faithful adornment property is lost.
□

As an aside, bcf is not faithful with respect to bCf (C can be sandwiched between f and c) and bCf is not faithful with respect to bcf (c can be sandwiched between c and b). As a result, each can pass some restrictions that the other cannot.

Proposition 5.3 *The bcf adornment class does not have the faithful adornment property.* □

The c adornment of the bcf class captures the existence of independent conditions on an argument, forgetting the nature of the conditions. There are situations when the actual condition on a c adorned argument, along with the built-in subgoals in a rule, enables us to deduce conditions that cannot be obtained otherwise. As an example, let $g1 = p(X)\ \&\ X < 10$, and $g2 = p(X)\ \&\ X > 10$ be two goals for the rule

$$(r): \quad p^c(X) :- Z > X\ \&\ q^{cf}(X, Z).$$

With goal $g1$, Z is free in the run time subgoal for q, and the adornment is accurate. However, with $g2$, a condition, $Z > 10$ can be deduced, and the subgoal is capturable by a stronger q^{cc} adornment. Faithfulness is thereby lost.

It is hoped that such cases are rare, so that the bcf class will usually be accurate. In essence, the bcf class is faithful if we do not deduce new conditions from the conditions in the goal and the conditions in the body of the rule:

Proposition 5.4 *The bcf adornment class has the faithful adornment property if new conditions are not deduced by a process of logical deduction from the given goal conditions and the conditions in a rule body.* □

6 Related Work

The problem of passing restrictions more general than a binding to a set of constants has been the subject of some other recent research. Ramakrishnan ([Ram88]) introduced Magic Templates to pass restrictions due to the presence of function symbols and the relationships between otherwise free arguments. However, the method generates non-range-restricted rules from range-restricted programs, and therefore cannot be applied in most database systems. Also, as we remarked in Section 5, the Bf adornment class used in [Ram88] is not faithful and is less descriptive than the bcf adornment class. Where non-range-restricted programs are acceptable, the Templates approach can benefit from the use of bcf class.

Balbin, Kemp, Meenakshi and Ramamohanarao ([BKMR89]) propose a folding/unfolding algorithm to push conditions into recursions. While our objectives are similar, our approach differs in many important ways. Balbin et al. assume that an adorned program (using the bf adornments) is given as input, and rewrite the program with the conditions pushed into lower strata. However, a bf adornment done without regard to conditions can cause their algorithm to fail, and their algorithm can benefit from our extension to the bcf adornment pattern. Our GMT algorithm offers an alternative to their algorithm for pushing conditions. For many programs (such as the program P in Example 1.1), the results of the two algorithms are similar, assuming we use the bcf adornments in both algorithms. However, there are cases where their algorithm is not applicable, while GMT is, and other cases where GMT has better behavior. For example, their algorithm cannot push conditions from built-in literals in a rule body into recursive subgoals, such as the condition $W > V$ on p^{cf} in rule $P1$ of Example 4.3; GMT clearly can. GMT has better behavior when conditions are pushed into a common subexpression. Their algorithm replicates the rules of the common predicate for each condition pushed, while GMT only replicates the grounding subgoals of the common predicate. There are also cases (grounding by recursive subgoals) where GMT, as presented here, is not applicable, while their algorithm is. However, as we noted in Section 4.2.1, GMT could be extended to allow grounding by recursive subgoals.

7 Conclusions

In this paper we have shown that the magic-sets techniques can be extended to deal with conditions. We have integrated (and partially implemented) an Extended Magic-sets algorithm into the rewrite optimization phase of the SQL-based Starburst prototype DBMS at IBM Almaden Research Center ([MFPR90]). The implementation also handles duplicates, grouping and aggregation operators of SQL ([MPR90]). We believe that the work described in this paper is important not only because it extends the theoretical scope of the magic-sets algorithm but also because it helps demonstrate that the magic-sets technique can be useful in practical relational database systems, particularly those with the power of SQL [ISO88].

We have extended the idea of bf adornments and shown that other adornment classes can be defined, opening up new ways to describe passing of information from larger classes of restrictions. We have presented mechanisms that let us (1) compare the descriptiveness of various adornment classes, (2) determine whether an adornment class can accurately predict the nature of run time goals, and (3) determine whether an adornment class can adorn a program better than another adornment class.

8 Acknowledgements

We thank Katherine Morris and Jeffrey D. Ullman for discussions on adornments. The Starburst project at IBM Almaden Research Center provided a stimulating environment for this work. Ashish Gupta and Yatin Saraiya provided helpful comments on drafts of this paper.

References

[BKMR89] Isaac Balbin, David B. Kemp, Krishnamurthy Meenakshi, and Kotagiri Ramamohanarao. `Propagating Constraints in Recursive Deductive Databases`. In *North American Conference on Logic Programming (NACLP), Cleveland, Ohio*, October 16-20 1989.

[BMSU86] Francois Bancilhon, David Maier, Yehoshua Sagiv, and Jeffrey D. Ullman. `Magic Sets and other Strange Ways to Implement Logic Programs`. In *Proceedings of the Fifth Symposium on Principles of Database Systems (PODS)*, pages 1–15, ACM SIGACT-SIGMOD-SIGART, March 1986.

[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, ACM SIGACT-SIGMOD-SIGART, March 1987.

[HFLP89] Laura M. Haas, J. C. Freytag, Guy M. Lohman, and Hamid Pirahesh. Extensible Query Processing in Starburst. In *Proceedings of ACM SIGMOD 1989 International Conference on Management of Data, Portland, OR*, pages 377–388, May 1989.

[HP88] Waqar Hasan and Hamid Pirahesh. Query Rewrite Optimization in Starburst. Research Report, RJ 6367 (62349), IBM Research Division, Computer Science, Almaden Research Center, San Jose, California 95120-6099, August 4 1988.

[ISO88] ISO_ANSI. Working Draft; Database Language SQL2. 1988.

[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, Atlantic City, NJ*, pages 247–258, May 23-25 1990.

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

[Mor88] Katherine A. Morris. An Algorithm for Ordering Subgoals in NAIL!. In *Proceedings of the Seventh Symposium on Principles of Database Systems (PODS)*, pages 82–88, ACM SIGACT-SIGMOD-SIGART, March 1988.

[Ram88] Raghu Ramakrishnan. Magic Templates: A Spellbinding Approach to Logic Programs. In Robert A. Kowalski and Keneth A. Bowen, editors, *Logic Programming: Proceedings of the Fifth International Conference and Symposium, Seattle, Vol 1*, pages 140–159, MIT Press, Cambridge, MA, August 1988.

[Ull88] Jeffrey D. Ullman. Principles of Database and Knowledge-Base Systems, Volume 1. Computer Science Press, 1988.