PUSHING CONSTRAINT SELECTIONS*

DIVESH SRIVASTAVA AND RAGHU RAMAKRISHNAN

▷ Bottom-up evaluation of a program-query pair in a constraint query language often computes only ground facts. Constraints do not contribute to answers, but are used only to prune derivations. The Magic Templates evaluation cannot utilize all the constraint information present in such program-query pairs while computing only ground facts. In general, constraint facts are computed, making the resulting evaluation more expensive.

We describe an optimization that propagates constraints occurring in the program and the query, such that the rewritten program fully utilizes the constraint information present in the original program. Only *constraint-relevant* facts are computed, and if the evaluation of the original program computed only ground facts, so does the evaluation of the rewritten program. Our procedure can be combined with the Magic Templates transformation to propagate query binding information in addition to the constraint information. We show that it is always better to defer the application of Magic Templates. ◁

1. INTRODUCTION

Recently, there have been attempts ([2, 4, 7, 11], among others) to increase the expressive power of database query languages by integrating constraint paradigms with logic-based database query languages. Such languages are referred to as *constraint query languages* (CQLs). Constraint programming paradigms are inherently declarative. Evaluating such programs can be expensive due to the manipula-

*This work was supported by a David and Lucile Packard Foundation Fellowship in Science and Engineering, a Presidential Young Investigator Award, with matching grants from Digital Equipment Corporation, Tandem, and Xerox, and NSF grant IRI-9011563. A preliminary version of this paper appeared in the *Proceedings of the ACM Symposium on Principles of Database Systems*, San Diego, California, June 1992.

Address correspondence to D. Srivastava, Computer Sciences Department, University of Wisconsin, Madison, WI 53706. Email:{divesh,raghu}@cs.wisc.edu.

Received December 1991; accepted March 1993.

THE JOURNAL OF LOGIC PROGRAMMING

655 Avenue of the Americas, New York, NY 10010 0743-1066/93/$6.00

tion of constraints, and hence optimizing such programs is very important. We consider the following problem: How can we optimize a CQL program-query pair $\langle P, Q \rangle$ by propagating constraints occurring in P and Q? More precisely, the problem is to find a set of constraints for each predicate such that the following statements hold:

- Adding the corresponding set of constraints to the body of each rule defining a predicate yields a program P' such that $\langle P, Q \rangle$ is query equivalent to $\langle P', Q \rangle$ (on all input EDBs).
- Only facts that are *constraint-relevant* to $\langle P, Q \rangle$ are computed in a bottom-up fixpoint evaluation of $\langle P', Q \rangle$ on an input EDB.

Constraint sets that satisfy the first condition are called query-relevant predicate (QRP) constraints; those that satisfy both conditions are called minimum QRP constraints.[1] The notion of *constraint relevance* is introduced to capture the information in the constraints present in P and Q. (We note that *every* fact for a predicate that appears in P or Q is constraint-relevant if neither P nor Q contains constraints.) Identifying and propagating QRP constraints is useful in two distinct situations:

- Often it is possible to evaluate queries on CQL programs without actually generating constraint facts [e.g., $p(X, Y; X \leq Y)$]. The constraints in the program are used to prune derivations, and only ground facts are generated.
- Even when constraint facts are generated, we may ensure termination in evaluating queries on CQL programs that would not have terminated if these constraints had not been propagated.

In this paper, we present a procedure that generates and propagates minimum QRP constraints (if it terminates) based on the definition and uses of program predicates (Section 4). By propagating minimum QRP constraints to the original program, we obtain a program that fully utilizes the constraint information present in the original program. This procedure is based on two subprocedures:

1. Procedure `Gen_Prop_predicate-constraints`, which generates and propagates constraints that are satisfied by program predicates based on their *definitions*.
2. Procedure `Gen_Prop_QRP-constraints`, which generates and propagates constraints based on the *uses* of program predicates, using fold/unfold transformations [14] and constraint manipulation.

We also show that determining whether (any representation for) the minimum QRP constraint for a predicate is a finite constraint set is undecidable (Section 3). We describe a class of programs for which this problem is decidable (Section 5). For this class of programs, our procedure for computing minimum QRP constraints always terminates.

The Magic Templates transformation [10] has been widely studied for propagating bindings. An important question is how procedures `Gen_Prop_predicate-`

[1] There is indeed a minimum QRP constraint, as we show later. Because we treat a constraint as equivalent to its set of ground instances, this definition is independent of the exact representation of the constraint set.

`constraints` and `Gen_Prop_QRP-constraints` interact with the use of Magic Templates.[2] Our results are as follows:

1. We present an algorithm based on Magic Templates followed by a finite sequence of fold/unfold transformations that essentially mimics the algorithm of [9], Section 6. This enables us to view the results of this paper, [1] and [9] in a uniform framework; namely, a combination of Magic Templates and (possibly simpler versions of) procedures `Gen_Prop_predicate-constraints` and `Gen_Prop_QRP-constraints`, in some order.
2. We examine various orderings in which procedures `Gen_Prop_predicate-constraints` and `Gen_Prop_QRP-constraints` and Magic Templates can be applied, and show that it is always better to defer the application of Magic Templates (Section 7).

1.1. Motivating Examples

Let us describe some examples that motivate the paper.

Example 1.1 (Computing flights). Consider the following program P:

$r1$: *cheaporshort*(S, D, T, C) $:-$ *flight*(S, D, T, C), $T \leq 240$.
$r2$: *cheaporshort*(S, D, T, C) $:-$ *flight*(S, D, T, C), $C \leq 150$.
$r3$: *flight*$(Src, Dst, Time, Cost)$ $:-$ *singleleg*$(Src, Dst, Time, Cost)$, $Cost > 0$, $Time > 0$.
$r4$: *flight*(S, D, T, C) $:-$ *flight*$(S, D1, T1, C1)$, *flight*$(D1, D, T2, C2)$, $T = T1 + T2 + 30$, $C = C1 + C2$.,

where *cheaporshort* is the query predicate, and *Cost* and *Time* fields in *singleleg* are values drawn from the reals. Although P is a CQL program, each fact computed in the bottom-up evaluation of P is just a tuple of constants; no constraint fact is computed. Given a query, for instance,

Query: ?-*cheaporshort*(*madison*, *seattle*, *Time*, *Cost*).,

one would like to compute only relevant *flight* facts: Clearly, *flight* facts that have a cost > \$150 and take more than 240 minutes are *not relevant* to answering this query and, hence, need not be computed in answering the query.

Magic Templates [10] is an important rewriting strategy that seeks to restrict the computation of facts that are irrelevant to answering a query. The Magic Templates rewriting of the preceding program P can take one of two approaches.

1. Use the constraints in the bodies of rules such as $r1$ to restrict computation of magic facts. For instance, the magic rule obtained from $r1$ (with *bbff* adornment for *cheaporshort*) could be

$$mr1: m\_flight(S, D, T) :- m\_cheaporshort(S, D), T \leq 240.$$

[2] If only the Magic Templates transformaton is used to optimize CQL programs, this could lead to the generation of constraint facts, even when the evaluation of the original program generates only ground facts. A motivation for this work, as for [1] and [9], is to take advantage of the constraints present in the program to reduce the potentially relevant facts computed, and yet compute only ground facts during the bottom-up evaluation of the rewritten program.

The (bottom-up) evaluation of this rule would require computing constraint facts. For instance, given the preceding query, the fact *m_flight* (*madison*, *seattle*, T; $T \leq 240$) would be computed using rule $mr1$. Using such constraint facts in a bottom-up evaluation is likely to be more expensive than using only ground facts in the evaluation.

2. One can compute only ground facts safely in the magic program by *not* making use of the constraints in the body of rule $r1$ to restrict computation of magic facts. For instance, the magic rule obtained from $r1$ (again with *bbff* adornment for *cheaporshort*), in this case, would be

   $mr1'$: *m_flight*(S, D) :- *m_cheaporshort*(S, D).

   The bottom-up evaluation of the magic program that includes rule $mr1'$ would compute many irrelevant facts because not all available constraints are made use of in the magic program. In particular, given the query

   Query: ?-*cheaporshort*(*madison*, *seattle*, *Time*, *Cost*).,

   one could compute many *flight* facts with cost > \$150 and taking more than 240 minutes; these facts are not relevant to answering the query.

The rewriting techniques proposed by Balbin et al. [1] and Mumick et al. [9] would not be able to optimize this program. The rewriting scheme proposed in this paper propagates the constraints in the bodies of rules $r1$ and $r2$ in the foregoing program into the definition of *flight*, as described in Example 4.3. The bottom-up evaluation of the rewritten program computes only ground facts. Further, given *any* query on *cheaporshort* (i.e., any pattern of bound arguments), the bottom-up evaluation of the rewritten program does not compute any *flight* fact with (*Cost* > 150)&(*Time* > 240).

Example 1.2. (Computing backward Fibonacci). Consider the following program P_{fib} to compute the Fibonacci numbers:

$r1$: *fib*$(0, 1)$.
$r2$: *fib*$(1, 1)$.
$r3$: *fib*$(N, X1 + X2)$:- $N > 1$, *fib*$(N - 1, X1)$, *fib*$(N - 2, X2)$.

This program can be queried with:

Query: ?-*fib*$(N, 5)$.

Using complete left-to-right sips, the Magic Templates transformation would transform P_{fib} (and the foregoing query) to P_{fib}^{mg}:

$r1$: *fib*$(0, 1)$:- *m_fib*$(0, 1)$.
$r2$: *fib*$(1, 1)$:- *m_fib*$(1, 1)$.
$r3$: *fib*$(N, X1 + X2)$:- *m_fib*$(N, X1 + X2)$, $N > 1$, *fib*$(N - 1, X1)$, *fib*$(N - 2, X2)$.
$r4$: *m_fib*$(N - 1, X1)$:- *m_fib*$(N, X1 + X2)$, $N > 1$.
$r5$: *m_fib*$(N - 2, X2)$:- *m_fib*$(N, X1 + X2)$, $N > 1$, *fib*$(N - 1, X1)$.
$r6$: *m_fib*$(N, 5)$.

A semi-naive bottom-up fixpoint evaluation of P_{fib}^{mg} computes facts as shown in Table 1. The answer $N = 4$ to the query is computed in the seventh iteration, but the evaluation does not terminate. Note that this evaluation generates constraint

TABLE 1. Derivations in a bottom-up evaluation of P^{mg}_{fib}.

Iteration	Derivations made
0	$\{r6{:}m\_fib(N1, 5)\}$
1	$\{r4{:}m\_fib(N1, V1; N1 > 0)\}$
2	$\{r2{:}fib(1, 1), \mathbf{r4{:}m\_fib(N1, V1; N1 > 0)}\}$
3	$\{r5{:}m\_fib(0, V2), \mathbf{r5{:}m\_fib(0, 4)}\}$
4	$\{r1{:}fib(0, 1)\}$
5	$\{r3{:}fib(2, 2)\}$
6	$\{r3{:}fib(3, 3), \mathbf{r5{:}m\_fib(1, V2)}, \mathbf{r5{:}m\_fib(1, 3)}\}$
7	$\{r3{:}fib(4, 5), \mathbf{r5{:}m\_fib(2, V2)}, \mathbf{r5{:}m\_fib(2, 2)}\}$
8	$\{r3{:}fib(5, 8), \mathbf{r5{:}m\_fib(3, V2)}, \mathbf{r5{:}m\_fib(3, 0)}\}$

facts for the magic predicate, *m_fib*. Subsumed facts are shown in boldface; these are discarded, and are not used to make new derivations.

We show how to propagate constraint information present in this program such that the bottom-up evaluation of the rewritten program always terminates, while computing all the answers to the query (Example 4.4).

2. PRELIMINARIES

We assume familiarity with the syntax and semantics of constraint logic programs, as well as the issues involved in the bottom-up evaluation of such programs (see [6] and [7] for details). We describe the Magic Templates transformation in Appendix B. A few important definitions are given here.

Definition 2.1 (Linear arithmetic constraint). A *linear arithmetic constraint* is of the form

$$a_1 X_1 + \cdots + a_n X_n \; op \; a_{n+1},$$

where $a_1, \ldots, a_{n+1}$ are real-valued coefficients of real-valued variables X_i, $1 \leq i \leq n$, and the operator *op* is one of $<$, $>$, $\leq$, $\geq$, and $=$.

In this paper, we consider only constraint query languages with *linear arithmetic constraints* and programs without negation. However, our techniques easily extend to programs with other types of constraints as well.

A *rule* is of the form

$$p(\bar{X}) :- C, p_1(\bar{X_1}), \ldots, p_n(\bar{X_n}).$$

where C is a conjunction of constraints and $p(\bar{X}), p_1(\bar{X_1}), \ldots, p_n(\bar{X_n})$ are literals. $p(\bar{X})$ is referred to as the *head* of the rule and $C, p_1(\bar{X_1}), \ldots, p_n(\bar{X_n})$ is referred to as the *body* of the rule. A rule with no body literals (though it could have a conjunction of constraints in the body) is referred to as a *constraint fact*. It is also represented as $p(\bar{X}; C)$, and is thus a conjunction of constraints [7, 10]; it is a finite representation of the (potentially) infinite set of ground facts that satisfy the conjunction of constraints C.

A *relation* is a finite set of such facts, i.e., a disjunction of conjunctions of constraints. A *database* is a finite set of relations. A *program* is a finite set of rules, and the meaning of a program is given by its least model [6].

Definition 2.2 (Derivation tree). Given a program P with database D, derivation trees in $\langle P, D \rangle$ are defined for ground facts as follows:

- Every ground instance h of a fact in D is a derivation tree for itself, consisting of a single node with label h.
- Let r be a rule

  $$p(\bar{X}) :- C, p_1(\overline{X_1}), \ldots, p_n(\overline{X_n}).$$

  in P, let d_i, $1 \leq i \leq n$, be facts with derivation trees T_i, and let θ be the *mgu* of $(p_1(\overline{X_1}), \ldots, p_n(\overline{X_n}))$ and $(d_1, \ldots, d_n)$, such that $C[\theta]$ is satisfiable. Then the following is a derivation tree for each ground instance $p(\bar{a})$ of $p(\bar{X})[\theta]$: the root is a node labeled with $p(\bar{a})$ and r, and each T_i, $1 \leq i \leq n$, is a child of the root.

Constraints in rules are viewed as conditions that determine whether or not a candidate tree is indeed a derivation tree; constraints are not themselves part of a tree.

Bottom-up evaluation of a program in a CQL proceeds by starting with the constraint facts in the database and repeatedly applying all the rules of the program, in iterations, to compute new constraint facts. The evaluation terminates once we have reached a fixpoint. We now intuitively describe a rule application, the basic step in a bottom-up evaluation. Consider a program rule

$$r: p(\bar{X}) :- C, p_1(\overline{X_1}), \ldots, p_n(\overline{X_n}).,$$

where r is just a label we use and is not part of the syntax of a rule. A derivation of a p fact using rule r consists of two steps:

- First, choose one p_i fact that is an instance of literal $p_i(\overline{X_i})$, for each $1 \leq i \leq n$, to obtain a satisfiable conjunction of constraints over the variables present in the body of rule r.
- Next, variables not present in the head of the rule are eliminated using variable (quantifier) elimination techniques to obtain a conjunction of constraints over the variables in $\bar{X}$.

This newly generated p fact must be compared against previously generated p facts to check whether it is indeed a new fact. An *application* of rule r consists of making all possible derivations that can be made using rule r and the set of facts known at the end of the previous iteration. If no new facts are computed in an iteration, the computation has reached a fixpoint.

Note that bottom-up evaluation uses the representation of the constraint facts *directly*, instead of working with the potentially infinite set of ground facts represented by the constraint facts. The equivalence of the constraint facts computed in a bottom-up evaluation of a program P and the meaning of P in terms of its least model is in terms of the ground facts represented by the constraint facts.

Theorem 2.1. Consider a program P and database D in a CQL with arithmetic constraints, and let $\mathcal{F}$ be the set of constraint facts computed in a bottom-up evaluation of $\langle P, D\rangle$. Let $\mathcal{M}$ be the meaning of $\langle P, D\rangle$ in terms of its least model. Then,

- *(soundness) each ground instance f of a constraint fact $F \in \mathcal{F}$ is in $\mathcal{M}$, and*
- *(completeness) each fact f in $\mathcal{M}$ is a ground instance of a constraint fact $F \in \mathcal{F}$.*

PROOF. Consider a program P and database D in a CQL with arithmetic constraints. Jaffar and Lassez [6] described a functional semantics for $\langle P, D\rangle$ in terms of an immediate consequence operator $T_{P,D}$ and showed that the least fixpoint of $T_{P,D}$, given by $T^{\omega}_{P,D}$, is equivalent to the meaning of $\langle P, D\rangle$ in terms of its least model. Let $gr(\mathcal{F})$ be the ground facts represented by the set of constraint facts $\mathcal{F}$.

Claim 1. Consider a set of (constraint) facts, D_1, for database and derived predicates of a program P. Let $\mathcal{F}_1$ be the set of constraint facts computed from D_1 by a single application of each rule in P. Let $T_P(gr(\mathcal{D}_1))$ be the set of ground facts obtained by a single application of the immediate consequence operator T_P on the set of ground facts, $gr(\mathcal{D}_1)$. Then, (1) each ground instance f_1 of a constraint fact $F_1 \in \mathcal{F}_1$ is in $T_P(gr(\mathcal{D}_1))$ and (2) each fact f_1 in $T_P(gr(\mathcal{D}_1))$ is a ground instance of a constraint fact $F_1 \in \mathcal{F}_1$.

The proof of Claim 1 follows from the decision procedure of Tarski [15] for the theory of real closed fields. All the operations needed for a rule application have straightforward analogues in Tarski's decision procedure; projection (on the variables of the head of a rule) corresponds to quantifier elimination, for instance.

The soundness and completeness results for the program can be shown by induction on the iterations of the bottom-up evaluation of $\langle P, D\rangle$. The database D provides the base case. Claim 1 provides the induction step. □

Consequently, with each constraint fact F computed by a bottom-up evaluation of the program, we can associate the set of derivation trees for each ground instance of F.

Given a program-query pair $\langle P, Q\rangle$, we can treat the query Q as the body of a rule defining a new predicate q, not occurring in P. The arity of q is the same as the number of variables in Q. The predicate q can now be treated as the query predicate, queried with all its arguments free. Whenever a query is given, we assume this transformation has been done and the query is treated as just another program rule.

Definition 2.3 (Constraint set). A *constraint set* is a disjunction of conjunctions (DNF) of constraints.

A constraint set $C_1(X_1, \ldots, X_n)$ is said to *imply* another constraint set $C_2(X_1, \ldots, X_n)$, denoted $C_1(X_1, \ldots, X_n) \supset C_2(X_1, \ldots, X_n)$, if whenever we substitute constant a_i for the variable X_i, $1 \le i \le n$, such that $C_1(a_1, \ldots, a_n)$ simplifies to **true**, then so does $C_2(a_1, \ldots, a_n)$.[3] This can be naturally extended to the case when C_1 and C_2 do not contain the same variables.

[3]Note that the use of the $\supset$ symbol is different from its traditional use as the superset operator.

For example, the conjunction $(X + Y \leq 4) \& (X \geq 2)$ implies $Y \leq 2$. The techniques described in [13] can be used to check for implication of constraint sets of linear arithmetic constraints.

Definition 2.4 (Predicate constraints). Given a CQL program P, a *predicate constraint* on a predicate p is a constraint set satisfied by each p fact that is derived during the bottom-up evaluation of P, independent of the facts in the EDB predicates.

Given a program P, a predicate constraint C_p on predicate p is said to be *minimal* if there does not exist a C_p', such that (1) C_p' is also a predicate constraint on p, (2) $C_p' \supset C_p$, and (3) $C_p \not\supset C_p'$. The existence of a unique *minimum* predicate constraint is guaranteed as a consequence of the following proposition.

Proposition 2.2. Given a program P, if constraint sets $C1_p$ and $C2_p$ are predicate constraints on p, then so is $C1_p \& C2_p$ (after conversion to DNF).

Definition 2.5 (Constraint relevance). Given a CQL program P with query predicate q, consider the complete set S of derivation trees that are associated with query answers, for every possible extension of the EDB predicates and every possible query on the query predicate.

A ground program fact $p(\bar{a})$ is said to be constraint-relevant to the query predicate if it occurs in at least one derivation tree in S. A constraint fact $p(\bar{X}; C)$ is said to be *constraint-relevant* to the query predicate if each ground instance of it is constraint-relevant to the query predicate.

Definition 2.6 (Query-relevant predicate constraints). Given a CQL program P, a *query-relevant predicate constraint* (QRP constraint) on a predicate p is a constraint set satisfied by each p fact that is derived during the bottom-up evaluation of P, independent of the facts in the EDB predicates, and which is constraint-relevant to a query predicate of P.

A proposition similar to Proposition 2.2 guarantees the existence of minimum QRP constraints.

In representing constraints on argument positions of a predicate, we use $\$i$ for the ith argument. Because constraints in rules are in terms of the variables (X, Y, etc.) in the rule, whereas predicate constraints and QRP constraints are in terms of argument positions (\$1, \$2, etc.), we need functions to convert between the two forms. We use $PTOL(p(\bar{X}), C)$ to convert a constraint set over the argument positions of p to an "equivalent" constraint set over the variables in $\bar{X}$.

Definition 2.7 [PTOL( )]. Consider a predicate p of arity n and a constraint set C on the argument positions of p. Let $p(\bar{X})$ be a literal, such that $\bar{X}$ is a tuple of n (not necessarily distinct) variables.

We define $PTOL(p(\bar{X}), C)$ as the constraint set obtained from C by replacing each argument position by the "corresponding" variable in that position in $p(\bar{X})$.

For example, if *flight* is a predicate of arity 4, then $PTOL(flight(S, D, T, C), (\$3 \leq 240) \vee (\$4 \leq 150))$ is given by $(T \leq 240) \vee (C \leq 150)$.

Similarly, we use $LTOP(p(\bar{X}), C)$ to convert a constraint set over the variables in $\bar{X}$ to an "equivalent" constraint set over the argument positions of p.

Definition 2.8 [LTOP( )]. Consider a literal $p(\bar{X})$, such that $\bar{X}$ is a tuple of n (not necessarily distinct) variables and a constraint set $C(\bar{X})$ on the variables in $\bar{X}$.

If $\bar{X}$ is a tuple of n distinct variables, we define $LTOP(p(\bar{X}), C(\bar{X}))$ as the constraint set obtained from $C(\bar{X})$ by replacing each variable in $\bar{X}$ by the "corresponding" argument position of the variable in $p(\bar{X})$.

If $\bar{X}$ is not a tuple of n distinct variables, we define $LTOP(p(\bar{X}), C(\bar{X}))$ as $LTOP(p(\bar{Y}), \Pi_{\bar{Y}}(C(\bar{X}) \& C_1(\bar{X}, \bar{Y})))$, where $\bar{Y}$ is a tuple of n distinct variables, distinct from the variables in $\bar{X}$, and $C_1(\bar{X}, \bar{Y})$ is a conjunction of equality constraints that equates each variable in $\bar{Y}$ with the variable in the corresponding position in $\bar{X}$. The $\Pi_{\bar{Y}}$ operation is the projection (quantifier elimination) operation and it guarantees that we focus attention in the constraint set on the variables in $\bar{Y}$.

For example, $LTOP(flight(S, D, T, C), (T \leq 240) \vee (C \leq 150))$ is given by $(\$3 \leq 240) \vee (\$4 \leq 150)$.

3. PROPAGATING CONSTRAINTS: THE PROBLEM

Definition 3.1 (Program core). Consider a program P in a constraint query language. The *core* of P is the program obtained from P by deleting all constraints in program rules.

Consider a CQL program P, with query predicate q, and let P_{core} be the core of the program P. In this section, we look at the problem of optimizing P by propagating constraints occurring in P to the bodies of rules in P_{core}. The intuition behind not altering the core of P while propagating constraints is that the core represents the syntactic structure of the program, which encodes the programmer's knowledge about the problem.

More precisely, we address the problem of finding a constraint set for each predicate such that:

1. Adding the constraint set for predicate p to the body of each rule[4] defining p yields a program P' such that P is query equivalent to P' (on all input EDBs).
2. The bottom-up evaluation of P' on an input EDB should compute only program facts such that each of these facts is constraint-relevant to the query predicate in P.

A rewritten program P' is said to be completely optimized with respect to the constraints present in the program P if it satisfies the preceding two conditions. In terms of the definition of QRP constraints, a rewritten program P' is said to be completely optimized with respect to the constraints present in P if each ground instance of each p fact computed in a bottom-up fixpoint evaluation of P' on an input EDB satisfies the minimum QRP constraint on p. We are interested in computing QRP constraints, because we make the assumption (as is common in optimization of database query languages) that query optimization should be independent of the facts in the EDB predicates.

[4] If the constraint set has more than one disjunct, this would mean creating copies of the rule, each copy containing one of the disjuncts. This is required because only conjunctions of constraints are allowed in the body of a rule.

Theorem 3.1. Given a CQL program P with linear arithmetic constraints, determining whether any representation for the minimum predicate constraint for a predicate p is a finite constraint set is undecidable.

PROOF. We first show that a variant of the safety (or finiteness) problem for logic programs is undecidable. Consider the Sebelik and Stepanek [12] reduction that showed that every partial recursive function can be expressed as a logic program P_{pr} with one constant symbol and one unary function symbol.

Now consider a program P_1 that has the rules

$$p(a) :- q(\bar{X}).$$
$$p(f(X)) :- p(X).$$

in addition to all the rules in P_{pr}, where p does not occur in P_{pr}, a is the only constant symbol in P_{pr}, f is the only function symbol in P_{pr}, and $q(\bar{X})$ is a query on P_{pr}.

Clearly, $p(a)$ is in the model for p iff $q(\bar{X})$ is satisfiable. If $p(a)$ is in the model, then so are $p(f(a)), p(f(f(a))), \ldots$, and the model of p in this program is infinite. Hence, the model of p is infinite iff $q(\bar{X})$ is satisfiable in P. Now, satisfiability of a query $q(\bar{X})$ on P_{pr} is undecidable, by reduction of the "halting problem" for partial recursive functions. Hence, it is undecidable whether the model of p is finite in this program.

We now reduce this problem to the problem of deciding whether any representation for the minimum predicate constraint for a predicate is finite. Intuitively, this reduction takes a logic program P and reduces it to a CQL program P' such that there is a unique representation for the minimum predicate constraint for predicate p in P', and the number of disjuncts in the minimum predicate constraint is finite iff the model of p in P is finite.

Consider a logic program P defining a unary predicate p [in addition to defining other (possibly) nonunary predicates]. Let there be just one constant, say a, and one unary function symbol, say f, appearing in P. (The foregoing program P_1 satisfies these conditions.) We transform the logic program P into a CQL program P' as follows:

- We replace all occurrences of the constant a by the numeric constant 0 in P'.
- All occurrences of $f(X)$ (appearing in the head or the body of a rule) in P are replaced by Y (a variable not appearing in the rule), and the conjunction of constraints $(X \geq 0) \& (Y = X + 2)$ is included in the body of the modified rule in P'.

It can be seen easily that there is a one-to-one correspondence between p facts in the model of p in P and p facts in the model of p in P'. Further, each p fact in the model of p in P' is a ground fact, with an even-valued argument ≥ 0. Hence, the minimum predicate constraint for p in P' is the possibly infinite disjunction

$$\bigvee_{i \geq 0} (\$1 = 2 * i).$$

Again, there is a one-to-one correspondence between p facts in the model of p in P' and the disjuncts in the minimum predicate constraint for p in P'. It can be

seen easily that this is a unique representation for the minimum predicate constraint for p in P', given the representations we consider for constraint sets.

Consequently, the number of disjuncts in the minimum predicate constraint for p in P' is finite iff the number of p facts in the model of p in P is finite. The undecidability result follows. □

In general, a constraint set with linear arithmetic constraints can have many different (though equivalent) representations. Even if a specific representation of the minimum predicate constraint is infinite, it *does not* follow that the minimum predicate constraint is infinite, because it could have an equivalent finite representation. The result is hence *independent of* any specific representation of the minimum predicate constraint.

The proof of Theorem 3.1 can also be used to show the following proposition.

Proposition 3.2. Given a CQL program P, where the only constraints allowed are of the form $X \leq c$, $X \geq c$, and $X \leq Y + c$, determining whether any representation for the minimum predicate constraint for a predicate p is a finite constraint set is undecidable.

Brodsky and Sagiv [3] show that it is undecidable whether a specific procedure for computing minimum predicate constraints computes finite predicate constraints. Proposition 3.2 does not follow from their result because they do not address the issue of multiple representations.

Theorem 3.3. Given a CQL program P with linear arithmetic constraints, determining whether any representation for the minimum QRP constraint for a predicate p is a finite constraint set is undecidable.

PROOF. Consider a CQL program P defining predicate p. Let q be a predicate not occurring in P. We add a new rule:

$$r{:}q(\bar{X}){:}\text{-}p(\bar{X}).$$

to P, where q has the same arity as p, and $\bar{X}$ is a tuple of distinct variables. Let q be the query predicate of the modified program P_1. Because r is the only rule defining q, every p fact is constraint-relevant to the query predicate q. Consequently, the minimum QRP constraint for p in P_1 is the same as the minimum predicate constraint for p in P.

From Theorem 3.1, we know that determining whether the minimum predicate constraint for p in P is a finite constraint set is undecidable. Hence, determining whether the minimum QRP constraint for predicate p in P_1 is a finite constraint set is also undecidable. □

As a corollary, we have the following proposition.

Proposition 3.4. Given a CQL program P, where the only constraints allowed are of the form $X \leq c$, $X \geq c$, and $X \leq Y + c$, determining whether any representation for the minimum QRP constraint for a predicate p is a finite constraint set is undecidable.

In Section 5, we describe a class of constraint query languages for which it is decidable whether the minimum QRP constraint for a predicate in a program can be represented as a finite constraint set.

Independent of its use in establishing Theorem 3.3, Theorem 3.1 is of interest because our procedure for generating (and propagating) minimum QRP constraints first generates (and propagates) minimum predicate constraints (see Section 4.4).

4. THE TRANSFORMATION: PROPAGATING CONSTRAINTS

In this section, we describe a rewriting procedure for propagating constraints in a CQL program. Our procedure has two components to it:

1. For each derived predicate p of a program P, it generates QRP constraints on p, using semantic properties of constraints.
2. It then uses the fold/unfold transformations [14] to propagate the QRP constraint on p into the program rules defining p.

Procedure `Gen_Prop_QRP-constraints` in Appendix C describes this procedure algorithmically. If the rewriting procedure terminates, it propagates QRP constraints for each derived predicate in the program, while preserving the core of the program. However, the rewriting procedure does not terminate in general.

A key feature of our rewriting procedure is that it makes essential use of semantic properties of constraints, unlike previous techniques that had a similar objective [1, 9]. As a consequence, we are able to optimize a larger class of programs than previous techniques.

4.1. An Example

First, we give a simple example of how the fold/unfold transformations can be used along with semantic properties of constraints to propagate constraint selections in a program.

Example 4.1. Consider the following program P with query predicate q:

$$r1: q(x) :- p_1(X,Y), p_2(Y), X+Y \le 6, X \ge 2.$$
$$r2: p_1(X,Y) :- b_1(X,Y).$$
$$r3: p_2(X) :- b_2(X).$$

First, two new rules are created:

$$r4: p_1'(X,Y) :- X+Y \le 6, X \ge 2, p_1(X,Y).$$
$$r5: p_2'(Y) :- Y \le 4, p_2(Y).$$

The body of each rule includes a derived literal from the body of rule $r1$, and the projection of the constraints in the body of $r1$ onto the variables of the derived literal. (Such a projection can be performed using the techniques described in [8].) Thus, although $Y \le 4$ is not present in the body of $r1$, it is implied by the conjunction of constraints $(X+Y \le 6) \& (X \ge 2)$.

Next, the definitions of p_1 and p_2 are unfolded[5] into the definitions of p_1' and p_2' to obtain

$$r4': p_1'(X,Y) :- X+Y \le 6, X \ge 2, b_1(X,Y).$$
$$r5': p_2'(Y) :- Y \le 4, b_2(Y).$$

[5]The fold/unfold transformations used are described in Appendix A.

Finally, rules $r4$ and $r5$ are folded into rule $r1$ to obtain

$$r1': q(X) :- p_1'(X,Y), p_2'(Y), X + Y \leq 6, X \geq 2.$$

The transformed program obtained (after deleting rules not reachable from the query predicate q) is P':

$$r1': q(X) :- p_1'(X,Y), p_2'(Y), X + Y \leq 6, X \geq 2.$$

$$r4': p_1'(X,Y) :- X + Y \leq 6, X \geq 2, b_1(X,Y).$$

$$r5': p_2'(Y) :- Y \leq 4, b_2(Y).$$

Note that P' is equivalent to P on the query predicate, the bottom-up evaluation of P' computes only ground facts, if the bottom-up evaluation of P does so, and the bottom-up evaluation of P' computes, in general, fewer facts than the bottom-up evaluation of P. Further, this program can now be rewritten using Magic Templates to take advantage of any constants in the actual query; if the sips use the bound-if-ground rule (i.e., the sips treat an argument as bound only if it is bound to a ground term), the Magic Templates transformed program also computes only ground facts.

Note that the *LTOP* of the conjunction of constraints in the body of $r4'$, viz. $(\$1 + \$2 \leq 6) \& (\$1 \geq 2)$, is the minimum QRP constraint for p_1 in the original program P. Similarly, the *LTOP* of the constraint in the body of rule $r5'$, viz. $\$1 \leq 4$, is the minimum QRP constraint for p_2 in the original program P. We have thus generated and propagated the minimum QRP constraints for the various derived predicates in the original program.

Neither the C transformation of [1] nor the GMT transformation of [9] would be able to propagate all the constraints in this example. Because the C transformation of [1] treats constraints as any other literal, it would not be able to propagate any constraints into the definition of p_2. The problem is that in rule $r1$, there is no explicit constraining literal on Y. Our technique utilizes the fact that $((X + Y \leq 6) \& (X \geq 2)) \supset (Y \leq 4)$, a semantic property of the conjunction of linear arithmetic constraints in the body of rule $r1$, to propagate constraints into the definition of p_2, and hence restrict the potentially relevant p_2 facts computed.

As described in [9], the GMT transformation does not handle constraints with arithmetic function symbols such as $+$. Consequently, it would not be able to propagate constraints either.

4.2. Generation of QRP Constraints

Our procedure `Gen_QRP-constraints`, described in Appendix C, for generating QRP constraints works iteratively. In each iteration, given "approximate" QRP constraints on each predicate defined in program P, the procedure computes a new approximation for the QRP constraints for each predicate in the program. The procedure terminates when a "fixpoint" is reached. Theorem 4.2 shows that, in the limit, procedure `Gen_QRP-constraints` does compute QRP constraints for each predicate in the program.

Nonrecursive Inference. Consider a rule r of the form:

$$r: p(\bar{X}) :- C_r(\bar{Y}), p_1(\overline{X_1}), \ldots, p_n(\overline{X_n}).,$$

where $C_r(\bar{Y})$ is the conjunction of constraints in the body of rule r. Given a constraint set C_p on the arguments of the head predicate p of rule r, a *literal constraint* on $p_i(\overline{X_i})$ in the body of r is a constraint set (on the variables in $\overline{X_i}$) that needs to be satisfied by each p_i fact that can be used [in literal $p_i(\overline{X_i})$] to derive a p fact (using rule r) that satisfies C_p.

Proposition 4.1. Consider a CQL program P, with linear arithmetic constraints. Given a rule r of the form

$$r: p(\bar{X}) :- C_r(\bar{Y}), p_1(\overline{X_1}), \ldots, p_n(\overline{X_n}).,$$

where $C_r(\bar{Y})$ is the conjunction of constraints in the body of rule r, if C_p is the desired constraint set on head predicate p,

$$C_{p_i(\overline{X_i})}(\overline{X_i}) = \Pi_{\overline{X_i}}\left(PTOL\left(p(\bar{X}), C_p\right) \& C_r(\bar{Y})\right)$$

is a literal constraint on $p_i(\overline{X_i})$ in the body of rule r.

Recursive Inference. Because each fact in the query predicate q is constraint-relevant to an answer to some query, procedure `Gen_QRP-constraints` initially assumes the constraint **true** as the "approximate" QRP constraint for predicate q, and the constraint **false** as the "approximate" QRP constraint for every other predicate defined in P. Procedure `Gen_QRP-constraints` works iteratively.

In each iteration, given the "approximate" QRP constraints on each predicate p as the desired constraint set on the head of each rule defining p, the procedure computes literal constraints $C_{p_i(\overline{X_i})}$ for each derived literal in the body of each rule defining p. Let $p(\overline{X_1}), \ldots, p(\overline{X_k})$ be all the occurrences of p in the bodies of the rules in P and let $C_{p(\overline{X_1})}, \ldots, C_{p(\overline{X_k})}$ be the corresponding literal constraints inferred. Let $C1_p$ be the "approximate" QRP constraint on p before the iteration and let $C2_p$ be $\bigvee_{i=1}^{k} LTOP(p(\overline{X_i}), C_{p(\overline{X_i})})$, i.e., the disjunction of the *LTOP*s of the various literal constraints inferred on p literals in this iteration. For each predicate p, the new "approximate" QRP constraint is given by $C1p \vee C2_p$. (Before adding disjuncts to the "approximate" QRP constraint, we can eliminate redundant disjuncts.) If, for each predicate p defined in P, $C1_p \equiv C1_p \vee C2_p$, procedure `Gen_QRP-constraints` has reached a fixpoint and it terminates; else, the procedure continues iterating with the new "approximate" QRP constraints.

Theorem 4.2. Given a CQL program P with linear arithmetic constraints, the constraint set generated for each derived predicate p in P by procedure `gen_QRP-constraints` *is a QRP constraint for p.*

PROOF. Consider a CQL program P with linear arithmetic constraints. For each derived predicate p in P, let C_p be the constraint set generated by procedure `gen_QRP-constraints`. We prove the result by contradiction.

Let us suppose that there exists some query Q, some database D, and some ground fact $p(\bar{a})$ in the least model of $\langle P, D \rangle$ that *does not* satisfy C_p and that occurs in a derivation tree for an answer to the query Q.

We associate a number with each program fact as follows: Consider the set of derivation trees $\mathcal{T}_Q$ for answers to query Q, and the set of all facts $p(\bar{a})$ that occur in at least one derivation tree in $\mathcal{T}_Q$ such that p is defined in P:

- If $p(\bar{a})$ is the root of a derivation tree T in $\mathcal{T}_Q$, then $p(\bar{a})$ is given the number 0.
- Else, let $p(\bar{a})$ have a parent $p_1(\overline{a_1})$ in a derivation tree T in $\mathcal{T}_Q$. Then $p(\bar{a})$ is given the number $j + 1$, where j is the lowest numbered parent of $p(\bar{a})$.

Choose any fact $p(\bar{a})$ that occurs in a derivation tree T in $\mathcal{T}_Q$, such that $p(\bar{a})$ does not satisfy C_p, and let its number be k.

Let $C_p^0, C_p^1, \ldots$ be the sequence of constraint sets generated by procedure `Gen_QRP-constraints` as "approximate" QRP constraints for each predicate p. We now show by induction that $p_2(\overline{a_2})$ facts with number j satisfy $C_{p_2}^j$.

For the base case, a fact $p_2(\overline{a_2})$ with number 0 is the root of a derivation tree T in $\mathcal{T}_Q$. It is an answer to the query and trivially satisfies $C_{p_2}^0$, which is **true**. For the induction step, assume that all p_2 facts with number $j \geq 0$ satisfy $C_{p_2}^j$, and consider fact $p_2(\overline{a_2})$ with number $j + 1$. This fact has as a parent $p_1(\overline{a_1})$ (and rule r), where $p_1(\overline{a_1})$ has number j. Now consider the rule r [with head $p_1(\overline{X_1})$] and the literal $p_2(\overline{X_2})$, where the fact $p_2(\overline{a_2})$ is used to compute $p_1(\overline{a_1})$. Given the constraint set $C_{p_1}^j$ [satisfied by $p_1(\overline{a_1})$ by hypothesis] on the head of rule r, fact $p_2(\overline{a_2})$ has to satisfy the *LTOP* of the literal constraint on $p_2(\overline{X_2})$, and hence $C_{p_2}^{j+1}$. This is because "projection" (or quantifier elimination) of linear arithmetic constraint sets can be done exactly using the algorithm described in [8], for instance. This concludes the induction. Hence, fact $p(\bar{a})$ satisfies C_p^k (because its number is k) and, hence, C_p. This contradicts the original assumption that $p(\bar{a})$ does not satisfy C_p, and concludes the proof of the theorem. □

Procedure `Gen_QRP-constraints` may not terminate, in general. Termination could be guaranteed by various modifications of our procedure. For example, instead of iterating until the QRP constraints "stabilize," we could iterate for a (predetermined) fixed number of iterations. If after the fixed number of iterations, the "approximate" QRP constraints have not stabilized, our procedure can return **true** (which is trivially a QRP constraint, though not the minimum possible) as the QRP constraint for program predicates. Clearly, this does not affect the correctness of our procedure. The larger the number of iterations chosen, the larger the class of programs for which we will be able to infer nontrivial QRP constraints. Clearly, there is a tradeoff here: how large a class of programs we wish to optimize versus the cost we are willing to incur in optimizing such programs. What bound to choose depends on the relative costs and is outside the scope of this paper.

4.3. Propagating QRP Constraints

If procedure `Gen_QRP-constraints` terminates (with the QRP constraint for p having m disjuncts), we can use the fold/unfold transformations, described in Appendix A, to propagate this QRP constraint into rules defining p. The propagation consists of three steps:

1. Perform a *definition* step creating m rules with head $p'(\overline{X})$ and the sole body literal being $p(\overline{X})$. Further, each of the m rules contains one of the m disjuncts of the QRP constraint generated for p.

2. Unfold the definition of p into each of the rules defining p'.
3. Fold the original definition of p' into each rule containing an occurrence of p.

With this, the QRP constraints generated for p have been propagated into the rules defining p. Procedure `Gen_Prop_QRP-constraints` in Appendix C describes this algorithmically. Example 4.1 illustrates this procedure for a simple program.

The correctness of the fold, unfold, and definition steps ensures the following statement.

Theorem 4.3. Given a CQL program P with linear arithmetic constraints, if procedure `Gen_Prop_QRP-constraints` *terminates, the rewritten program is equivalent to the original program with respect to the query predicates, on all input EDBs.*

The following result indicates that if the original program could be evaluated "efficiently," so can the rewritten program.

Theorem 4.4. Consider a CQL program P with linear arithmetic constraints, such that the bottom-up evaluation of a program P on database D computes only ground facts. Then:

- *The bottom-up evaluation of the rewritten program P', obtained using procedure* `Gen_Prop_QRP-constraints`, *on database D also computes only ground facts.*
- *The bottom-up evaluation of P' on database D computes a subset of the facts computed by the bottom-up evaluation of P on database D.*
- *If the QRP constraint* C_p *generated for each derived predicate p were such that the intersection of no two disjuncts of* C_p *was satisfiable, the bottom-up evaluation of P' on database D makes a subset of the derivations made by the bottom-up evaluation of P on database D.*

PROOF. Consider a program P such that the bottom-up evaluation of P on database D computes only ground facts. We prove the results by considering the sequence of definition, unfold, and fold steps performed by procedure `Gen_Prop_QRP-constraints`.

First, consider the definition step. Each new rule is of the form

$$r: p'(\bar{X}) :- C, p(\bar{X}).,$$

where $\bar{X}$ is a tuple of distinct variables and C is a conjunction of constraints involving only the variables in $\bar{X}$. (It is the *PTOL* of a disjunct in the QRP constraint C_p generated for predicate p.) Because each rule defining p computes only ground facts, adding an additional conjunction of constraints C only prevents the computation of certain facts; it does not affect the property that all p' facts computed are ground. Also, the set of facts computed for p' is a subset of the set of facts computed for p because of the additional conjunction of constraints C in the body of rule r. Further, if C_p were such that the intersection of no two disjuncts was satisfiable, each of the rules defining p' would compute distinct facts. Hence, the set of derivations made for p' is a subset of the set of derivations made for p.

Next, consider the unfold step. The definition of p is unfolded into the rules defining p'. This does not affect the set of p' facts computed nor the derivations made.

Finally, consider the fold step. The fold step essentially replaces occurrences of p in rule bodies by p'. Because the set of facts computed for p' is a subset of the set of facts computed for p, by replacing occurrences of p by p' in the body of a rule defining p_1, no new facts are computed for p_1 by this rule. Because the facts computed for p_1 prior to the fold step were ground, so are facts computed for p_1 subsequent to this fold step. If the derivations made by each rule defining p' were distinct, the derivations made for p_1 subsequent to the fold step does not increase. This completes the proof of the theorem. □

4.4. *Using Inferred Predicate Constraints to Obtain QRP Constraints*

The QRP constraints generated and propagated by procedure `Gen_Prop_QRP-constraints` need not be the *minimum* QRP constraints, in general. However, if the program P is of a certain form, then we *can* guarantee that procedure `Gen_Prop_QRP-constraints` does generate and propagate the minimum QRP constraints (if it terminates). Theorem 4.7 provides conditions on the form of such programs. We first describe a program where the minimum QRP constraint is not generated using procedure `Gen_QRP-constraints`.

Example 4.2. Consider the following program P:

$$r1: q(X,Y) :- a(X,Y), X \leq 10.$$
$$r2: a(X,Y) :- p(X,Y), Y \leq X.$$
$$r3: a(X,Y) :- a(X,Z), a(Z,Y).,$$

where q is the query predicate. Assuming **true** as the QRP constraint for q, we would generate the literal constraint $X \leq 10$ for the occurrence of $a(X,Y)$ in the body of rule $r1$. Now, assuming the constraint set $\$1 \leq 10$ as the "approximate" constraint set for a, we would generate the following literal constraints: For $a(X,Z)$ in rule $r3$, we would get $X \leq 10$; for $a(Z,Y)$ in rule $r3$, we would get **true**. Procedure `Gen_QRP-constraints` would infer **true** (or, unconstrained) as the QRP constraint for a and the procedure would terminate.

Note, however, the following. Each a fact that is derived using rule $r2$ of program P has the constraint $\$2 \leq \$1$. These facts will be used in the recursive rule $r3$ to derive more a facts. If each of the facts used in the body of $r3$ satisfies $\$2 \leq \$1$, then so does the head fact derived using these facts and rule $r3$. Thus, each a fact derived using rules $r2$ and $r3$ in the preceding program P satisfies the predicate constraint $\$2 \leq \$1$. [Similarly, each q fact derived using rule $r1$ in the preceding program P satisfies the predicate constraint $(\$2 \leq \$1) \& (\$1 \leq 10)$.] If, for each a literal in the program, the *PTOL* of the predicate constraint $\$2 \leq \$1$ were explicitly introduced, we would get the following program $P1$:

$$r1': q(X,Y) :- a(X,Y), X \leq 10, Y \leq X.$$
$$r2: a(X,Y) :- p(X,Y), Y \leq X.$$
$$r3': a(X,Y) :- a(X,Z), Z \leq -X, a(Z,Y), Y \leq Z.$$

Now, introducing these constraints does not change the facts computed by the program $P1$. However, with program $P1$ we can use procedure `Gen_QRP-constraints` to obtain the minimum QRP constraint $((\$1 \leq 10) \& (\$2 \leq \$1))$ for a. This constraint can now be propagated using the fold/unfold transformations to reduce the number of facts computed by the rewritten program P'.

Given a CQL program P with linear arithmetic constraints, there is a procedure to enumerate minimum predicate constraints for program predicates. Intuitively, given constraints on base predicates, the procedure works by iteratively computing the constraints that hold on derived predicates bottom-up. Procedure `Gen_predicate-constraints` in Appendix C algorithmically describes this. In general, procedure `Gen_predicate-constraints` may not terminate.

Theorem 4.5. Given a CQL program P with linear arithmetic constraints, procedure `Gen_predicate-constraints` *generates the minimum predicate constraint for each derived predicate of the program P.*

PROOF. Consider a CQL program P. For each derived predicate p in P, let C_p be the constraint set generated by procedure `Gen_predicate-constraints`.

Claim 1. C_p is a predicate constraint for p.

PROOF OF CLAIM 1. We prove this by contradiction. Let us suppose that there exists some database D and some ground fact $p(\bar{a})$ that *does not* satisfy C_p and that occurs in a derivation tree.

We associate a number with each program fact as follows: Consider the set of derivation trees $\mathcal{T}$ and the set of all facts $p(\bar{a})$ that occur in at least one derivation tree in $\mathcal{T}$.

- Let $p(\bar{a})$ be a leaf node in some derivation tree T in $\mathcal{T}$. Then $p(\bar{a})$ is given the number 0.
- Else, let $p(\bar{a})$ be such that in some derivation tree T in $\mathcal{T}$, a child of $p(\bar{a})$ is of the form $p_1(\overline{a_1})$. Then $p(\bar{a})$ is given the number $j+1$, where j is the highest numbered child of $p(\bar{a})$.

Choose any fact $p(\bar{a})$ such that $p(\bar{a})$ does not satisfy C_p, and let its number be k.

Let $C_p^0, C_p^1, \ldots$ be the sequence of constraint sets generated by procedure `Gen_predicate-constraints` as "approximate" predicate constraints for predicate p. We now show by induction that $p_2(\overline{a_2})$ facts with number j satisfy $C_{p_2}^j$.

For the base case, a fact $p_2(\overline{a_2})$ with number 0 is a database fact. It trivially satisfies C_p^0, the predicate constraint given on the database predicate. For the induction step, assume that all facts with number $j \geq 0$ satisfy $C_{p_2}^j$, and consider fact $p_2(\overline{a_2})$ with number $j+1$. There is a derivation tree and a node in the derivation tree with label $p_2(\overline{a_2})$ (and rule r) such that each child of it has a number $\leq j$. Now consider the rule r [with head $p_2(\overline{X_2})$] that is used to compute $p_2(\overline{a_2})$. Given the constraint set $C_{p_1}^j$ on each p_1 literal in the body of rule r, fact $p_2(\overline{a_2})$ has to satisfy the *LTOP* of the inferred head constraint on $p_2(\overline{X_2})$, and hence $C_{p_2}^{j+1}$. This is because quantifier elimination of linear arithmetic constraint sets can be done exactly. This concludes the induction. Hence, fact $p(\bar{a})$ satisfies C_p^k (because its number is k) and, hence, C_p. This contradicts the original assumption that $p(\bar{a})$ does not satisfy C_p, and concludes the proof of Claim 1. □

Claim 2. C_p is a *minimum* predicate constraint for p.

PROOF OF CLAIM 2. We prove this by contradiction. Let $C_p^0, C_p^1, \ldots$ be the sequence of constraint sets generated by procedure `Gen_predicate-constraints` as "approximate" predicate constraints for predicate p. Consider the lowest numbered C_p^k such that there exists a fact $p(\bar{a})$ that satisfies C_p^k and is not in the least model of $\langle P, D \rangle$, i.e., each fact that is an instance of $C_p^m, m < k$ is in the least model of $\langle P, D \rangle$. The number k cannot be 0, because C_p^0 is **false** for all derived predicate p and no p fact satisfies this constraint; C_p^0 is not **false** only for database predicates p, and for these predicates, we have assumed that the minimum predicate constraints were made available as input.

Because ground fact $p(\bar{a})$ satisfies C_p^k, it satisfies some disjunct of C_p^k that is not present in C_p^{k-1}. From procedure `Gen_predicate-constraints` we know that this disjunct was generated in iteration k of procedure `Gen_predicate-constraints` using some rule r. Consider this rule and the literals in the body of r. By hypothesis, each fact that satisfies $C_{p_1}^{k-1}$ is present in the least model and can be used in body literal $p_1(\overline{X_1})$. Because quantifier elimination of linear arithmetic constraint sets can be done exactly, each ground fact that satisfies the inferred head constraint C_p^k can also be derived and must be present in the least model. This contradicts the original assumption that $p(\bar{a})$ is not present in the least model, and concludes the proof of Claim 2, as well as the proof of the theorem. □

Brodsky and Sagiv [3] studied this problem of generating predicate constraints for a special case, where the only constraints allowed are of the form $\$i < \$j + c$. Van Gelder [17] also studied a similar but more restricted problem; the techniques of [17] can be used only to derive a single conjunction of constraints on the arguments of a predicate; general constraint sets (which are disjunctions of conjunctions) are not inferred.

Using procedure `Gen_predicate-constraints` for generating predicate constraints, one can infer that $\$2 \leq \$1$ is a predicate constraint for a in program P of Example 4.2. One can also infer that, on the *flight* predicate in program P of Example 1.1, $\$4 > 0$ (that is, the cost of each flight is > 0) as well as $\$3 > 0$ (that is, the time taken by each flight is > 0). For each of these examples, our procedure for generating predicate constraints terminates.

If procedure `Gen_predicate-constraints` terminates, we can associate the *PTOL* of the predicate constraint for p with body occurrences of p. Procedure `Gen_Prop_predicate-constraints` in Appendix C describes this algorithmically.

Theorem 4.6. Given a CQL program P with linear arithmetic constraints, if procedure `Gen_Prop_predicate-constraints` *terminates (resulting in program P'), then*:

- *The rewritten program P' is equivalent to the original program with respect to all derived predicates, on all input EDBs that satisfy the predicate constraints for the database predicates.*
- *If the bottom-up evaluation of P on database D computes only ground facts, the bottom-up evaluation of P' on database D computes only ground facts.*
- *If the bottom-up evaluation of P on database D computes only ground facts, and the predicate constraint* $\mathrm{C1_p}$ *generated for each derived predicate p were such*

that the intersection of no two disjuncts of $C1_p$ *was satisfiable, then the bottom-up evaluation of* P' *on database* D *makes the same set of derivations for each predicate* p *as the bottom-up evaluation of* P *on database* D.

The proof of Theorem 4.6 follows easily from the fact that procedure `Gen_predicate-constraints` generates minimum predicate constraints, and the form of the rules in the rewritten program obtained using procedure `Gen_Prop_predicate-constraints`.

One of the main results of this paper is the following theorem about when procedure `Gen_Prop_QRP-constraints` generates and propagates minimum QRP constraints.

Theorem 4.7. Consider a CQL program P *with linear arithmetic constraints. Let the minimum predicate constraint for each predicate* p_i *be a finite constraint set,* C'_{p_i}. *Further, let the constraints in the body of each rule in* P *imply the constraint set* $PTOL(p_i(\overline{X_i}), C'_{p_i})$, *for each literal* $p_i(\overline{X_i})$ *in the body of the rule. Then, if it terminates, procedure* `Gen_Prop_QRP-constraints` *generates and propagates the minimum QRP constraints for each derived predicate of the program.*

PROOF. Consider a CQL program P and let the constraints in the body of each rule in P imply the constraint set $PTOL(p_i(\overline{X_i}), C'_{p_i})$, for each literal $p_i(\overline{X_i})$ in the body of the rule, where C'_{p_i} is the minimum predicate constraint for predicate p_i. For each derived predicate p in P, let C_p be the constraint set generated by procedure `Gen_Prop_QRP-constraints`. From Theorem 4.2, we already know that it is a QRP constraint for p. We only need to show that it is the *minimum* QRP constraint for p.

We prove the result by contradiction. Because the constraints in the body of each rule in P imply the constraint set $PTOL(p_i(\overline{X_i}), C'_{p_i})$, for each literal $p_i(\overline{X_i})$ in the body of the rule, each ground fact $p(\bar{a})$ that satisfies C_p is present in the least model of $\langle P, D\rangle$. Hence, we only need to show that each such fact is constraint-relevant to an answer to the query Q.

Let $C_p^0, C_p^1, \ldots$ be the sequence of constraint sets generated by procedure `Gen_QRP-constraints` as "approximate" QRP constraints for predicate p. Consider the lowest numbered C_p^k such that there exists a fact $p(\bar{a})$ that satisfies C_p^k and is not constraint-relevant to an answer to the query Q. The number k cannot be equal to 0 because C_p^0 is **false** for all nonquery predicates and no p fact can satisfy this constraint; C_p^0 is **true** for the query predicate and every p fact has to satisfy this constraint.

Because ground fact $p(\bar{a})$ satisfies C_p^k, $k > 0$, it satisfies some disjunct of C_p^k that is not present in C_p^{k-1}. From procedure `Gen_QRP-constraints`, we know that this disjunct was generated in iteration k of procedure `Gen_QRP-constraints` using some rule r. Now consider the rule r and the literal $p(\overline{X})$, which was used to generate this disjunct. By hypothesis, the "approximate" QRP constraint $C_{p_1}^{k-1}$ for the head $p_1(\overline{X_1})$ is such that each ground fact that satisfies this constraint set is constraint-relevant to an answer to query Q.

Because quantifier elimination of linear arithmetic constraint sets can be done exactly, and ground fact $p(\bar{a})$ is in the least model, there is a derivation tree where $p(\bar{a})$ is a child of some $p_1(\overline{a_1})$ (with rule r) that satisfies C_{p_1} and is constraint-relevant to an answer to query Q. Consequently, $p(\bar{a})$ is also constraint-relevant to the same answer to query Q as $p_1(\overline{a_1})$ is. This contradicts the assumption that $p(\bar{a})$ is

not constraint-relevant to an answer to query Q and completes the proof of the theorem. □

4.5. *Putting it all Together*

Given a CQL program P with linear arithmetic constraints, and a query predicate q, our procedure for generating (and propagating) minimum QRP constraints has two components to it:

1. First, we use procedure `Gen_Prop_predicate-constraints` for generating and propagating minimum predicate constraints for each predicate in P.
2. Next, we use procedure `Gen_Prop_QRP-constraints` to generate and propagate the QRP constraints for each derived predicate in P.

Procedure `Constraint_rewrite` in Appendix C describes this algorithmically.

The following result follows from Theorems 4.5, 4.6, and 4.7.

Theorem 4.8. Given a CQL program P with linear arithmetic constraints, if procedure `Constraint_rewrite` *terminates, the QRP constraints generated and propagated are minimum QRP constraints for each derived predicate in P.*

Let us illustrate the result of applying this procedure to our original motivating program from Example 1.1.

Example 4.3 (Computing flights: rewritten and optimized). Consider the program P in Example 1.1.

$r1$: *cheaporshort*(S, D, T, C) $:-$ *flight*$(S, D, T, C), T \leq 240.$
$r2$: *cheaporshort*(S, D, T, C) $:-$ *flight*$(S, D, T, C), C \leq 150.$
$r3$: *flight*$(Src, Dst, Time, Cost)$ $:-$ *singleleg*$(Src, Dst, Time, Cost), Cost > 0,$ $Time > 0.$
$r4$: *flight*(S, D, T, C) $:-$ *flight*$(S, D1, T1, C1)$, *flight*$(D1, D, T2, C2)$, $T = T1 + T2 + 30, C = C1 + C2.$,

where *cheaporshort* is the query predicate. We first add a rule

$$r0{:}q_1(S, D, T, C) :- \textit{cheaporshort}(S, D, T, C).$$

and q_1 is the new query predicate. Our procedure for generating minimum predicate constraints terminates on this example, and concludes that the predicate *flight* has the minimum predicate constraint $(\$3 > 0) \& (\$4 > 0)$. It also concludes that the predicate *cheaporshort* has the minimum predicate constraint $((\$3 > 0) \& (\$3 \leq 240) \& (\$4 > 0)) \vee ((\$3 > 0) \& (\$4 > 0) \& (\$4 \leq 150))$. For each body predicate occurrence of *flight* and *cheaporshort*, we introduce the *PTOL* of the predicate constraints into the rule body.

Now, procedure `Gen_Prop_QRP-constraints` can be applied to obtain the disjunctive constraint $((\$3 > 0) \& (\$3 \leq 240) \& (\$4 > 0)) \vee ((\$3 > 0) \& (\$4 > 0) \& (\$4 \leq 150))$ as the minimum QRP constraint on *flight* as well as *cheaporshort*. Propagating these (minimum) QRP constraints and deleting the rules defining q_1 results

in the following program P':

$r1$: $cheaporshort(S, D, T, C)$ $:- flight'(S, D, T, C), T > 0, T \leq 240, C > 0.$

$r2$: $cheaporshort(S, D, T, C)$ $:- flight'(S, D, T, C), T > 0, C > 0, C \leq 150.$

$r3$: $cheaporshort(S, D, T, C)$ $:- flight'(S, D, T, C), T > 0, T \leq 240, C > 0, C \leq 150.$

$r4$: $flight'(Src, Dst, Time, Cost)$ $:- Time > 0, Time \leq 240, singleleg(Src, Dst, Time, Cost), Cost > 0.$

$r5$: $flight'(S, D, T, C)$ $:- T > 0, T \leq 240, C > 0, flight'(S, D1, T1, C1), flight'(D1, D, T2, C2), T1 > 0, T2 > 0, T = T1 + T2 + 30, C1 > 0, C2 > 0, C = C1 + C2.$

$r6$: $flight'(Src, Dst, Time, Cost)$ $:- Time > 0, Cost \leq 150, singleleg(Src, Dst, Time, Cost), Cost > 0.$

$r7$: $flight'(S, D, T, C)$ $:- T > 0, C > 0, C \leq 150, flight'(S, D1, T1, C1), flight'(D1, D, T2, C2), T1 > 0, T2 > 0, T = T1 + T2 + 30, C1 > 0, C2 > 0, C = C1 + C2.$

Note that the bottom-up evaluation of P' does not compute any *flight'* fact that is not constraint-relevant to the query predicate. In particular, no *flight'* fact with time > 240 minutes and cost > \$150 is computed. Further, each of the facts computed during a bottom-up evaluation of P' is a ground fact.

Given a query, such as

Query: ?-$cheaporshort(madison, seattle, Time, Cost)$.,

one could now rewrite this program using Magic Templates and the bound-if-ground rule for sips to take advantage of the pattern of constants in the actual query. This is *not* something that our optimization (based on generating minimum QRP constraints and propagating these into the bodies of rules) was designed to do. The magic rewritten program is now able to utilize the various constraints in the program rules, as well as the constants in the query, without computing constraint facts.

We next show how propagating predicate constraints can make the difference between nontermination and termination in answering queries on a CQL program, even when the evaluation computes constraint facts.

Example 4.4 (Computing backward Fibonacci: rewritten and optimized). Consider the program P_{fib} from Example 1.2:

$r1$: $fib(0, 1)$.

$r2$: $fib(1, 1)$.

$r3$: $fib(N, X1 + X2) :- N > 1, fib(N - 1, X1), fib(N - 2, X2)$.

Note that $\$2 \geq 1$ is a predicate constraint (though not the minimum) for *fib*. We can associate the *PTOL* of this constraint with each body occurrence of *fib* in rule $r3$ of P_{fib}. By introducing these constraints, we get the following program P_{fib_1}:

$r1$: $fib(0, 1)$.

$r2$: $fib(1, 1)$.

$r3$: $fib(N, X1 + X2) :- N > 1, fib(N - 1, X1), X1 \geq 1, fib(N - 2, X2), X2 \geq 1$.

The Magic Templates transformation of the resultant program is $P^{mg}_{fib\_1}$:

$r1$:$fib(0,1)$ $:- m\_fib(0,1).$
$r2$:$fib(1,1)$ $:- m\_fib(1,1).$
$r3$:$fib(N, X1+X2)$ $:- m\_fib(N, X1+X2), N>1, fib(N-1, X1), X1 \geq 1, fib(N-2, X2), X2 \geq 1.$
$r4$:$m\_fib(N-1, X1)$ $:- m\_fib(N, X1+X2), N>1, X1 \geq 1, X2 \geq 1.$
$r5$:$m\_fib(N-2, X2)$ $:- m\_fib(N, X1+X2), N>1, fib(N-1, X1), X1 \geq 1, X2 \geq 1.$
$r6$:$m\_fib(N, 5).$

A seminaive bottom-up fixpoint evaluation of $P^{mg}_{fib\_1}$ computes facts as shown in Table 2. Note that the answer to the query is computed in the seventh iteration, and the evaluation terminates after the eighth iteration because no new derivations are made during the eighth iteration. The additional constraints in the bodies of rules $r3$, $r4$, and $r5$ of $P^{mg}_{fib\_1}$ prevent subsequent derivations.

In a similar manner, given the query

Query: ?-$fib(N, 6)$.,

a seminaive bottom-up evaluation of $P^{mg}_{fib\_1}$ [with $r6$ replaced by $m\_fib(N, 6)$] terminates, and answers "no" because there is no N whose Fibonacci number is 6. The bottom-up evaluation of P^{mg}_{fib} would not terminate.

4.6. Issues in the Generation and Propagation of QRP Constraints

Given a CQL program P, procedure `Constraint_rewrite` generates and propagates minimum QRP constraints for each derived predicate in the program. Propagating QRP constraints into rule bodies has several advantages. For instance, in the bottom-up evaluation of program P' of Example 4.3:

1. Fewer *flight'* facts need be computed (because the constraints in the rules defining *cheaporshort* are used earlier). In particular, no *flight'* fact with time > 240 and cost > 150 is computed. Because there could be an arbitrary

TABLE 2. Derivations in a bottom-up evaluation of $P^{mg}_{fib\_1}$.

Iteration	Derivations made
0	{$r6$:$m\_fib(N1, 5)$}
1	{$r4$:$m\_fib(N1, V1; N1 > 0, V1 \geq 1, V1 \leq 4)$}
2	{$r2$:$fib(1,1)$, **r4:m_fib(N1, V1; N1 > 0, V1 ≥ 1, V1 ≤ 3)**}
3	{$r5$:$m\_fib(0, V2; V2 \geq 1, V2 \leq 3)$, $r5$:$m\_fib(0,4)$}
4	{r1:fib(0, 1)}
5	{r3:fib(2, 2)}
6	{r3:fib(3, 3), **r5:m_fib(1, 3), r5:m_fib(1, V2; V2 ≥ 1, V2 ≤ 2)**}
7	{r3:fib(4, 5), **r5:m_fib(2, 2), r5:m_fib(2, 1)**}
8	{ }

number of *flight* facts (in P) with time > 240 and cost > 150, considerable savings (in terms of the number of facts derived) are achieved.

2. These constraints can be used for effective indexing of relations. In particular, the constraints $Cost \leq 150$ and $Time \leq 240$ could be used to efficiently retrieve (via B trees, etc.) $singleleg(\_, \_, Time, Cost)$ tuples satisfying this constraint. This can improve the efficiency of rule application.

With disjunctive constraints, however, using fold/unfold transformations may lead to an increase in the number of derivations of each fact, though the number of facts computed may decrease. For instance, if the *singleleg* relation contained $singleleg(madison, chicago, 50, 100)$, the original program P in Example 4.3 would derive $flight(madison, chicago, 50, 100)$ just once using the nonrecursive rule, whereas $flight'(madison, chicago, 50, 100)$ would be derived twice using the nonrecursive rules in P'. Because the number of disjuncts in the minimum QRP constraint, though finite, is unbounded, the number of derivations could increase considerably, in general.

Notice that this problem of multiple derivations of *flight'* facts arises because the minimum QRP constraint for *flight* is a nontrivial disjunction and the two disjuncts $((\$3 > 0) \& (\$3 \leq 240) \& (\$4 > 0))$ and $((\$3 > 0) \& (\$4 > 0) \& (\$4 \leq 150))$ "overlap" in that their intersection is satisfiable. There are two possible solutions to this problem:

- First, one can represent the minimum QRP constraint in such a way that the intersection of no two disjuncts is satisfiable. The algorithms described in [13] can be used to obtain such nonoverlapping disjuncts. Thus, for instance, the minimum QRP constraint for *flight* can be represented as $((\$3 > 0) \& (\$3 \leq 240) \& (\$4 > 0) \& (\$4 \leq 150)) \vee ((\$3 > 0) \& (\$3 \leq 240) \& (\$4 > 0) \& (\$4 > 150)) \vee ((\$3 > 0) \& (\$3 > 240) \& (\$4 > 0) \& (\$4 > 150))$. If this representation of the minimum QRP constraint for *flight'* is propagated, the rewritten program does not make any more derivations than the original program. However, the number of rules in the rewritten program could increase exponentially, because representing a given constraint set C as a constraint set C' with nonoverlapping disjuncts could result in an exponential increase in the number of disjuncts.
- Another possible solution is to bound the number of disjuncts to one by simplification of the QRP constraint, using constraint manipulation techniques. This is always possible, though the result of such a simplification would be a nonminimal QRP constraint, in general. Thus, for instance, in program P in Example 4.3, bounding the number of disjuncts in the QRP constraint to one, results in obtaining the QRP constraint as $(\$3 > 0) \& (\$4 > 0)$. Propagating this would not result in any reduction in the number of *flight'* facts computed.

In practical terms, there is a trade-off between the number of potentially relevant facts computed, the number of derivations of potentially relevant facts, and the overheads of applying a large number of rules. What choice to make depends on estimates of the relative costs of evaluation of the alternative rewritten programs. We do not discuss this issue further because it is outside the scope of this paper.

5. SUFFICIENT CONDITIONS FOR TERMINATION

Given a CQL program P with linear arithmetic constraints, determining whether (any representation for) the minimum QRP constraint for a predicate p is a finite constraint set is undecidable, according to Theorem 3.3. However, one might be able to obtain decidability results if we restrict the classes of constraints that we consider in the CQL. In this section, we show that for a restricted class of constraint query languages we can obtain decidability results, and that our procedure for generating (and propagating) the minimum QRP constraints always terminates.

Example 5.1. Consider program $P1$ of Example 4.2:

$$r1{:}q(X,Y){:}-a(X,Y),X\leq 10,Y\leq X.$$
$$r2{:}a(X,Y){:}-p(X,Y),Y\leq X.$$
$$r3'{:}a(X,Y){:}-a(X,Z),Z\leq X,a(Z,Y),Y\leq Z.$$

It is easy to see that, because there are no arithmetic function symbols in the program and the only arithmetic predicate used is $\leq$, the only "simple" constraints that can be part of the QRP constraint for a, generated by procedure `Gen_QRP-constraints`, are $\$1 \leq 10$, $\$2 \leq 10$, $10 \leq \$1$, $10 \leq \$2$, $\$1 \leq \$1$, $\$1 \leq \$2$, $\$2 \leq \$2$, and $\$2 \leq \$1$. No constant other than 10 can be part of the QRP constraint generated for a, because that would require the use of an arithmetic function symbol to create the new constant.

Each disjunct in a constraint set can contain any or all of these "simple" constraints. Because there are 8 "simple" constraints, there can be at most $2^8 = 256$ disjuncts in the QRP constraint generated for a. Each iteration of our procedure to generate QRP constraints checks whether the "new" constraints are subsumed by the "approximate" QRP constraint, and adds at least one "new" disjunct in each iteration (else it terminates). Because there are only a bounded number (256) of disjuncts *possible*, our procedure can iterate only 256 times, before it *must* terminate.

In the case of $P1$, our procedure, `Gen_Prop_QRP-constraints`, terminates in just two iterations.

Theorem 5.1. Consider a constraint query language with linear arithmetic constraints of the form X op Y and X op c, where op is one of $\{\leq, \geq, \leq, >\}$ *and c is a constant. Given a CQL program P with these constraints, there is a terminating procedure to compute minimum QRP constraints.*

PROOF. Consider a CQL with linear arithmetic constraints of the form $X\ op\ Y$ and $X\ op\ c$, where op is one of $\{\leq, \geq, <, >\}$ and c is a constant. In such a constraint query language, no n-ary ($n > 0$) function symbols (such as $+$ or $*$) are permitted. A "simple" constraint (on the arguments of a predicate) in a program in such a CQL can either be of the form $\$i \leq c$, $\$i < c$, $c \leq \$i$, $c < \$i$, $\$i \leq \$j$, or $\$i < \$j$. (A simple constraint involving $\geq$ or $>$ can be treated as an equivalent constraint using $\leq$ or $<$.) If predicate p has arity k, there can be at most $2k^2 + 4k$ "simple" constraints that can be part of the predicate constraint or the QRP constraint generated for p. (There can be k^2 constraints each of the forms $\$i \leq \$j$ and

$\$i < \$j$, and k constraints each[6] of the forms $\$i \leq c$, $\$i < c$, $c \leq \$i$, and $c < \$i$.) Consequently, there can be at most 2^{2k^2+4k} disjuncts in the predicate constraint or the QRP constraint for p.

If program P contains n predicates, with arity at most k, it is easy to see that at most $n * 2^{2k^2+4k}$ disjuncts are possible in the predicate or the QRP constraints generated for predicates in P. Because each iteration of procedure `Gen_predicate-constraints` and procedure `Gen_QRP-constraints` adds at least one "new" disjunct, each of these procedures terminates in at most $n * 2^{2k^2+4k}$ iterations.[7] Consequently, Procedure `Constraint_rewrite` terminates on this class of programs, having generated and propagated minimum QRP constraints. This also gives us the decidability result.

This result can be generalized easily to constraint query languages with no n-ary ($n > 0$) function symbols and only a finite number of constraint predicate symbols.

6. UNDERSTANDING PREVIOUS TECHNIQUES

Balbin et al. [1] and Mumick et al. [9] consider the problem of propagating constraints such as $X > 10$ in a range-restricted, function-free CQL program P.[8] Both [1] and [9] use a combination of constraint propagation and Magic Templates.

A fundamental limitation of each of these techniques is that they do not utilize semantic properties of constraints. The techniques presented in this paper make essential use of such properties and are, hence, able to optimize programs that could not be handled by previous techniques.

6.1. Balbin et al.'s C Transformation

The approach taken by Balbin et al. [1] is to try to propagate constraints using (a more limited version than we consider of) fold/unfold and then apply Magic Sets. Given a program P, their technique is depicted in Figure 1. It can be split into three phases:

1. First, they have an adornment phase, which uses the *bf* adornment, where *b* stands for bound and *f* stands for free. The sips (sideways information passing strategies) selected treat an argument as bound only if it is bound to a ground term. Let the adorned program obtained be P^{ad}.
2. Second, they perform a C transformation on the adorned program. The C transformation is expressed as a sequence of fold, unfold, and definition steps using the fold/unfold transformations of Tamaki and Sato [14]. This step propagates constraints into the recursive rules, while obtaining a query-equivalent program. A constraint in a rule body is treated as any other rule body literal for the purposes of the transformation.[9] Let the C-transformed program be $P^{ad,C}$.

[6] Even if there are, say two constants c_1 and c_2 in the program, in each disjunct there can only be one of $\$i \leq c_1$ and $\$i \leq c_2$ present. The other constraint is redundant.

[7] This is a combinatorial upper bound for the number of iterations taken. For most programs, we expect the bound to be considerably lower.

[8] Range restrictedness is a sufficient syntactic condition for all the facts computed during the bottom-up evaluation of a program to be ground facts.

[9] Balbin et al. [1] refers to constraints in P as *constraining predicates*. For instance, $>$ is a constraining predicate and $X > 3$ is a constraining literal.

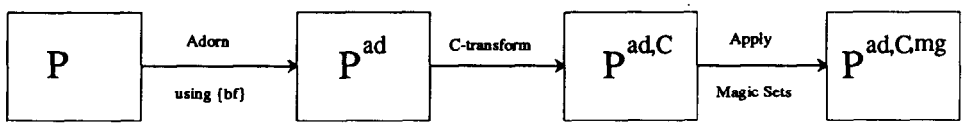

FIGURE 1. The transformation of Balbin et al.

3. In the third phase, they perform the standard Magic Sets transformation on the C transformed program. The resultant program is denoted $P^{ad,C,mg}$.

Thus, the approach of [1] is to C transform the (adorned) program before the Magic Sets rewriting is applied. Further, given a program P that has only range-restricted rules, the transformation of [1] has the property that each of P^{ad}, $P^{ad,C}$, and $P^{ad,C,mg}$ has only range-restricted rules. Consequently, all the facts computed during the bottom-up evaluation of the C transformed program as well as the magic rewritten program are ground facts.

Our procedures for generating and propagating minimum QRP constraints can be straightforwardly used to replace the constraint propagation phase of the technique of Balbin et al. [1]. The resulting technique can optimize a larger class of programs than [1].

6.2. Mumick et al.'s GMT Transformation

The approach taken by Mumick et al. [9] (the Ground Magic Set, or GMT, transformation) directly extends the Magic Templates rewriting of [10] to support propagation of arithmetic constraints without leading to computation of constraint facts. Although [9] presents the GMT transformation as a single algorithm that combines Magic Templates with the propagation of constraints, to understand the GMT transformation, it is best to think of GMT as a three step transformation, as shown in Figure 2:

1. Given a program P, the first step is an adornment phase. They generalize the class of bound (b) and free (f) adornments to include a condition (c) adornment that describes selections involving arithmetic inequalities. They describe how sips can be modified to allow conditions, in addition to bindings, to be passed sideways. Let the adorned program obtained be P^{ad}.
2. In the second step, they take a *bcf* adorned program, and apply the Magic Templates transformation of [10] to get a program that may have non-range restricted magic rules. Let the Magic Templates transformed program be $P^{ad,mg}$.
3. Finally, they *ground* the magic rules in $P^{ad,mg}$ to get a range restricted program $P^{ad,mg,gr}$.

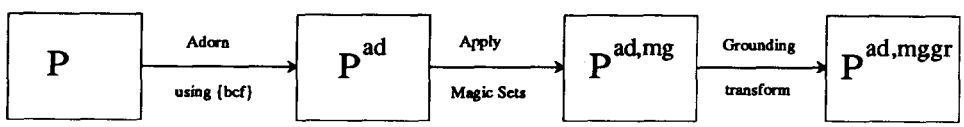

FIGURE 2. The transformation of Mumick et al.

Our contribution is to show that this final (and quite complicated) grounding step can be understood as a sequence of fold/unfold transformations using the system of Tamaki and Sato [14]. Thus, the technique of [9] can also be decomposed into a combination of the standard Magic Templates rewriting and the fold/unfold transformation, each of which is well understood.

Semantic properties of constraints can also be used to enhance the adornment phase in the GMT algorithm of [9] and permit a larger class of programs to be optimized.

In contrast to [1], the approach of [9] is to magic transform the (adorned) program *before* applying the fold/unfold transformations. This intermediate (magic) program $P^{ad,mg}$ could compute constraint facts. However, the final program obtained, $P^{ad,mg,gr}$, has only range-restricted rules and, hence, computes only ground facts. Mumick et al. [9] impose some conditions on programs on which the grounding transformation (and hence the GMT transformation) would work. We first describe these conditions, and for the class of programs that satisfy these conditions, we describe the sequence of fold/unfold transformations that "captures" the grounding step of [9].

Definition 6.1 (Groundable program). Consider a range-restricted *bcf* adorned program P^{ad}. The program P^{ad} is said to be *groundable* if for each predicate p^{*c*} in P^{ad} with at least one c in its adornment, and for each rule in P^{ad} defining p^{*c*}, the following holds: Each variable in a conditioned (c) argument position of the head of the rule also occurs in an ordinary body literal $q(\bar{X})$, such that q is not recursive with p^{*c*}.

Such a literal $q(\bar{X})$ is referred to as a *grounding subgoal.*

The main result of Mumick et al. [9] is contained in the following theorem.

Theorem 6.1. Given a range-restricted, groundable program P^{ad}, bcf adorned for the query Q, the program $P^{ad,mg,gr}$, obtained after the GMT transformation has the properties that (1) *$P^{ad,mg,gr}$ is range-restricted and* (2) *$P^{ad,mg,gr}$ is query equivalent to P^{ad} with respect to the query Q.*

We now describe the sequence of fold/unfold transformations that captures the "grounding" step in the GMT transformation. The fold/unfold transformations work on the rules of $P^{ad,mg}$ based on the structure of the strongly connected components (SCC) of P^{ad}. Procedure `Ground_Fold_Unfold` gives an algorithmic description of the fold/unfold transformations that capture the grounding step:

```
Ground_Fold_Unfold(P^ad, P^ad,mg)
{
  let S_1,...,S_n be a topological sorting of the SCCs of P^ad,
    with S_1 as the SCC of the query predicate.
  R = the rules of P^ad,mg.
  for i = 1 to n do
    Ground_SCC_Fold_Unfold (S_i, R) /* this procedure modifies
  R. */
  end for
  the resulting program is a range-restricted program.
```

Ground_SCC_Fold_Unfold ($S_i, \mathscr{R}$)
{
let S_i define predicates $p_{i1}^{*c*}, \ldots, p_{im}^{*c*}$.
/* these predicates are used only in SCCs $S_1, \ldots, S_i$ in P^{ad}. */
let $m\_p_{ij}^{*c*}$, $1 \le j \le m$ be the corresponding magic predicates.
let $R_{m\_p,ext}$ be the rules defining $m\_p_{ij}^{*c*}$ in $\mathscr{R}$ obtained from body occurrences of p_{ij}^{*c*} in SCCs $S_1, \ldots, S_{i-1}$.
let $R_{m\_p,int}$ be the rules defining $m\_p_{ij}^{*c*}$ in $\mathscr{R}$ obtained from body occurrences of p_{ij}^{*c*} in S_i.
let R_p be the rules defining p_{ij}^{*c*}, $1 \le j \le m$ in $\mathscr{R}$.
let R_{m_lower} be the rules defining m_q in $\mathscr{R}$, obtained from occurrences of q in S_i, where q is defined in SCC S_j, $j > i$.

Definition Step:
let G_k be the set of grounding subgoals (and associated constraints) in rule $r_{p,k}$ (defining p_{ij}^{*c*}) in R_p.
for each rule $r_{p,k}$ (defining p_{ij}^{*c*}) in R_p, create a new predicate s_k_p.
/* [MFPR90] refers to these as supplementary predicates. */
let the rule defining s_k_p be: $r_{s\_k\_p}: s\_k\_p() :- m\_p_{ij}^{*c*}(), G_k$.
/* this rule is range-restricted, since it contains G_k. */
let R_{defn} be all these new rules.

Unfold Step:
unfold the definition of $m\_p_{ij}^{*c*}$, $1 \le j \le m$ into R_{defn} and R_{m_lower}.
let R_{unf} be the resulting set of rules.
let $R_{m\_p,unf}$ be the subset of R_{unf} containing an occurrence of $m\_p_{ij}^{*c*}$, $1 \le j \le m$.
let $R_{no,unf}$ be the subset of R_{unf} containing no occurrence of $m\_p_{ij}^{*c*}$, $1 \le j \le m$.

Fold Step:
fold $r_{s\_k\_p}$ (the definition of s_k_p) into each rule in R_p and $R_{m\_p,unf}$.
let R_{fold} be the resulting set of rules.

the set of rules $R_{m\_p,ext}$, $R_{m\_p,int}$, R_p and R_{m_lower} is replaced by R_{fold} and $R_{no,unf}$.
/* since no rule defining $m\_p_{ij}^{*c*}$ is reachable from the query predicate */
}

We believe that procedure Ground_Fold_Unfold produces the same program as the grounding step of the GMT transformation. We do not provide a formal

proof of this claim, although the similarity in the two procedures should be evident. We do prove, however, that procedure `Ground_Fold_Unfold` results in range-restricted programs, and the resultant program is query-equivalent to the original, as is the result of [9].

Theorem 6.2. Given a range-restricted, groundable program P^{ad}, bcf adorned for the query Q, and the (possibly) non-range-restricted program $P^{ad,mg}$ obtained by the Magic Templates rewriting of P^{ad} using grounding sips [9], *the program P' obtained by using procedure* `Ground_Fold_Unfold` *$(P^{ad}, P^{ad,mg})$ is such that* (1) *P' is a range-restricted program and* (2) *P' is query equivalent to P^{ad} with respect to the query Q.*

PROOF. We prove the first result by induction on the SCC structure of P^{ad} based on the sequence of invocations of procedure `Ground_SCC_Fold_Unfold`. The proof of the second result follows from the correctness of the Magic Templates transformation and the correctness of the fold/unfold transformations used.

We use the following notation in this proof. Let $\mathscr{R}$ be the set of rules in the program prior to an invocation of procedure `Ground_SCC_Fold_Unfold` on SCC S_i. Let $p_{i1}^{*c*}, \ldots, p_{im}^{*c*}$ be the predicates defined in SCC S_i. Let $m\_p_{ij}^{*c*}$, $1 \le j \le m$, be the corresponding magic predicates. Let $R_{m\_p,ext}$ be the rules defining $m\_p_{ij}^{*c*}$ in $\mathscr{R}$ obtained from occurrences of p_{ij}^{*c*} in SCCs $S_1, \ldots, S_{i-1}$ of P^{ad}. Let $R_{m\_p,int}$ be the rules defining $m\_p_{ij}^{*c*}$ in $\mathscr{R}$ obtained from occurrences of p_{ij}^{*c*} in S_i. Let R_p be the rules in $\mathscr{R}$ defining p_{ij}^{*c*}. Let R_{m_lower} be the rules defining m_q in $\mathscr{R}$, obtained from occurrences of q in S_i, where q is defined in SCC S_j, $j > i$. Let R_{fold} and $R_{no,unf}$ be the rules replacing $R_{m\_p,ext}$, $R_{m\_p,int}$, R_p, and R_{m_lower} in $\mathscr{R}$ after the invocation of procedure `Ground_SCC_Fold_Unfold` on S_i.

We prove the result by proving the following claims:

Claim 1. In $\mathscr{R}$:

1. All rules defining nonmagic predicates are range-restricted.
2. Only magic predicates that have a '*c*' (condition adornment) in their adornment could have non-range-restricted rules defining it, i.e., all rules defining magic predicates adorned using only the *bf* adornment are range-restricted.
3. There is no occurrence of a magic predicate with a *c* in its adornment in the body of any rule in $R_{m\_p,ext}$.

PROOF OF CLAIM 1. For the base case, Claims 1.1 and 1.2 are true of $P^{ad,mg}$, because the *bcf* adornment ensures that only magic predicates that have a '*c*' adornment can have non-range-restricted rules defining it, and Claim 1.3 is true of the "seed" magic rule generated from the query.

For the induction step, assume the claim is true prior to an invocation of `Ground_SCC_Fold_Unfold` on SCC S_i. Now consider the rules $R_{m\_p,ext}$, $R_{m\_p,int}$, R_p, and R_{m_lower}. Because we consider only groundable programs, each rule $r_{p,k}$ in R_p is of the form

$$r_{p,k}: p_{ij}^{*c*}(\bar{X}) :- m\_p_{ij}^{*c*}(\overline{X_1}), G_k, N_k.,$$

where G_k is the set of grounding subgoals (and associated constraints) from the corresponding rule in P^{ad} and N_k is the set of remaining subgoals. The grounding

sips of [9] ensures that each grounding (nonrecursive) subgoal (and constraints on it) precedes each of the nongrounding subgoals (and corresponding constraints).

As a consequence, each rule in $R_{m\_p,int}$ (derived from $r_{p,k}$) is of the form

$$m\_p^{*c*}_{ij_2}(\overline{X_2}) :- m\_p^{*c*}_{ij}(\overline{X_1}), G_k, C_k, M_k.,$$

where G_k grounds each c variable in $\overline{X_1}$, C_k is a set of constraining literals, which could result in this rule being non-range-restricted, and M_k may contain some recursive subgoals of $r_{p,k}$. Note that each variable in a condition (c) position of the head has to occur in C_k, and each variable in C_k that is not "ground" subsequently occurs in a condition variable in the head.

First, consider the rules in $R_{no,unf}$. These are obtained by unfolding the rules in $R_{m\_p,ext}$ into the rules in R_{defn} and R_{m_lower}. The rules in $R_{no,unf}$ obtained from R_{m_lower} will be part of $R_{m\_p,ext}$ for SCC S_h, $h > i$. There is no occurrence of a magic predicate with a 'c' in its adornment in the body of any of these rules, because by the induction hypothesis, none of the rules in $R_{m\_p,ext}$ had any magic predicate with a 'c' in its adornment.

A rule in R_{defn} is of the form

$$s\_k\_p() :- m\_p^{*c*}_{ij}(\overline{X_2}), G_k,$$

where G_k grounds all conditioned variables in $\overline{X_2}$.

Consider the rules in $R_{no,unf}$ obtained from R_{defn}. Because each rule in R_{defn} is range-restricted and G_k grounds all conditioned variables in the $m\_p^{*c*}_{ij}$ literal, each rule in $R_{no,unf}$ obtained from R_{defn} is also range-restricted. Rules in $R_{no,unf}$ defining magic predicates with only *bf* adornments are obviously range-restricted.

Now, consider the rules in $R_{m\_p,unf}$. These are obtained by unfolding the rules in $R_{m\_p,int}$ into the rules in R_{defn} and R_{m_lower}. Given the form of a rule in $R_{m\_p,int}$, a rule in $R_{m\_p,unf}$ obtained from R_{defn} is of the form

$$s\_k\_p() :- m\_p^{*c*}_{ij_h}(\overline{X_h}), G_h, C_h, M_h, G_k.,$$

where G_h is the set of grounding subgoals for conditioned variables in $\overline{X_h}$. This rule is also range-restricted, because every variable in C_h is ground by G_k (the variables in C_h not ground by M_h are precisely those that appeared in a condition argument of the magic literal in the rule defining s_k_p prior to the unfolding). Rules in $R_{m\_p,unf}$ defining magic predicates with only *bf* adornments are obviously range-restricted.

Finally, consider the rules in R_{fold}. These are obtained by folding the original definition of s_k_p into rules in R_p and $R_{m\_p,unf}$. From the form of these rules (previously given), it can be seen that if there is a magic literal $m\_p^{*c*}_{ij}(\overline{X_1})$ present in the body of a rule in R_p or $R_{m\_p,unf}$, then the grounding subgoals G_k are also present in the body of that rule. Consequently, the folding step can be performed. This does not affect the range restrictedness and eliminates *all* occurrences of magic predicates $R_{m\_p_{ij}}$ with a 'c' in its adornment.

Thus, the only rules in $R_{no,unf}$ and R_{fold} that are not range-restricted are those derived from rules in R_{m_lower} that define a magic predicate with a 'c' in its adornment. This completes the induction step, and the proof of Claims 1.1 to 1.3. □

Claim 2. After an invocation of procedure `Ground_SCC_Fold_Unfold` on SCC S_i, there are no rules in $\mathscr{R}$ defining any m_p_{ij} that has a 'c' in its adornment, nor are there any body occurrences of such $m\_p_{ij}^{*c*}$ literals.

PROOF OF CLAIM 2. For the base case, this is trivially true prior to the invocation on SCC S_1. For the induction step, assume that this is true prior to the invocation of procedure `Ground_SCC_Fold_Unfold` on SCC S_i, and consider the ith invocation. In procedure `Ground_SCC_Fold_Unfold`, each body occurrence of m_p_{ij} with a 'c' in its adornment has been folded away because of the presence of associated grounding subgoals. Because no rule defining $m\_p_{ij}^{*c*}$ is reachable, each of these rules can be deleted from $\mathscr{R}$ without affecting query equivalence. This completes the induction step and the proof of Claim 2. □

From Claim 2, we have that after the invocation of procedure `Ground_SCC_Fold_Unfold` on SCC S_n, there are no rules in $\mathscr{R}$ defining any magic predicate m_p that has a 'c' in its adornment, nor is there any body occurrence of such an m_p^{*c*} literal. Hence, the only rules in $\mathscr{R}$ are rules defining nonmagic predicates (original adorned predicates and supplementary predicates) and magic predicates with *bf* adornments. From Claim 1, we have that all rules defining these predicates are range-restricted rules. This concludes the proof of the theorem. □

The key intuition behind why the GMT transformation results in range-restricted rules is that although the rules defining the magic predicates in $P^{ad,mg}$ are not range-restricted, one could eliminate the rules defining those predicates and replace them by auxiliary predicates all of whose rules are range-restricted.

We illustrate how the sequence of fold/unfold steps of procedure `Ground_Fold_Unfold` captures the "grounding" step using the following example.

Example 6.1 (Fold/unfold captures grounding step). Consider the following (adorned) program-query pair P from Example 4.3 in [9]:

$$r1{:}p^{cf}(X,Y) :- U > 10, q^{ccf}(X,U,V), W > V, p^{cf}(W,Y).$$
$$r2{:}p^{cf}(X,Y) :- u^{cf}(X,Y).$$
$$r3{:}q^{ccf}(X,Y,Z) :- q1^{cf}(X,U), q2^{fc}(W,Y), q3^{bbf}(U,W,Z).$$
$$\text{Query: ?-}X > 10, p^{cf}(X,Y).$$

For rule $r1$, the set of grounding subgoals is given by $\{U > 10, q^{ccf}(X,U,V)\}$, for rule $r2$, the set of grounding subgoals is given by $\{u^{cf}(X,Y)\}$, and for rule $r3$, the set of grounding subgoals is given by $\{q1^{cf}(X,U), q2^{fc}(W,Y)\}$.

Using the Magic Templates transformation of [10] and full left-to-right sips, we obtain P^{mg}:

$$mr1{:}m\_p^{cf}(X) \quad :- X > 10.$$
$$mr2{:}m\_p^{cf}(W) \quad :- m\_p^{cf}(X), U > 10, q^{ccf}(X,U,V), W > V.$$
$$r1{:}\ p^{cf}(X,Y) \quad :- m\_p^{cf}(X), U > 10, q^{ccf}(X,U,V), W > V, p^{cf}(W,Y).$$
$$r2{:}\ p^{cf}(X,Y) \quad :- m\_p^{cf}(X), u^{cf}(X,Y).$$
$$mr3{:}m\_q^{ccf}(X,U) \quad :- m\_p^{cf}(X), U > 10.$$
$$r3{:}\ q^{ccf}(X,Y,Z) \quad :- m\_q^{ccf}(X,Y), q1^{cf}(X,U), q2^{fc}(W,Y), q3^{bbf}(U,W,Z).$$

The fold/unfold transformations work on the SCC structure of P. Because there are two SCCs in P (the higher one defining p^{cf} and the lower one defining q^{ccf}), we have two iterations, first for the higher SCC and then for the lower SCC.

For the first iteration, we have $R_{m\_p,ext} = \{mr1\}$, $R_{m\_p,int} = \{mr2\}$, $R_p = \{r1, r2\}$, and $R_{m\_lower} = \{mr3\}$. In the first step for this iteration, we define two new predicates $s\_1\_p^{cf}$ and $s\_2\_p^{cf}$ using the following rules:

$$r4{:}s\_1\_p^{cf}(X,V) :- m\_p^{cf}(X), U > 10, q^{ccf}(X,U,V).$$

$$r5{:}s\_2\_p^{cf}(X,Y) :- m\_p^{cf}(X), u^{cf}(X,Y).$$

In the second step, we unfold the definition of m_p^{cf} (rules $mr1$ and $mr2$) into rules $r4$, $r5$, and $mr3$ to get

$$r41: \quad s\_1\_p^{cf}(X,V) :- X > 10, U > 10, q^{ccf}(X,U,V).$$

$$r42: \quad s\_1\_p^{cf}(X,V) :- m\_p^{cf}(X1), U1 > 10, q^{ccf}(X1,U1,V1), X > V1, U > 10, q^{ccf}(X,U,V).$$

$$r51: \quad s\_2\_p^{cf}(X,Y) :- X > 10, u^{cf}(X,Y).$$

$$r52: \quad s\_2\_p^{cf}(X,Y) :- m\_p^{cf}(X1), U1 > 10, q^{ccf}(X1,U1,V1), X > V1, u^{cf}(X,Y).$$

$$mr31: \quad m\_q^{ccf}(X,U) :- X > 10, U > 10.$$

$$mr32: \quad m\_q^{ccf}(X,U) :- m\_p^{cf}(X1), U1 > 10, q^{ccf}(X1,U1,V1), X > V1, U > 10.$$

In the third step of the first iteration, we fold rules $r4$ and $r5$ into the bodies of $r42$, $r52$, $mr32$, $r1$, and $r2$ to get

$$r43: \ s\_1\_p^{cf}(X,V) \quad :- s\_1\_p^{cf}(X1,V1), X > V1, U > 10, q^{ccf}(X,U,V).$$
$$r53: \ s\_2\_p^{cf}(X,Y) \quad :- s\_1\_p^{cf}(X1,V1), X > V1, u^{cf}(X,Y).$$
$$mr33{:}m\_q^{ccf}(X,U) \quad :- s\_1\_p^{cf}(X1,V1), X > V1, U > 10.$$
$$r11: \ p^{cf}(X,Y) \quad :- s\_1\_p^{cf}(X,V), W > V, p^{cf}(W,Y).$$
$$r21: \ p^{cf}(X,Y) \quad :- s\_2\_p^{cf}(X,Y).$$

The result of the first iteration is the set of rules $\{r41, r51, mr31, r43, r53, mr33, r11, r21, r3\}$.

For the second iteration, we now have $R_{m\_q,ext} = \{mr31, mr33\}$, $R_{m\_q,int} = \emptyset$, $R_q = \{r3\}$, and $R_{m\_lower} = \emptyset$. In the first step for this iteration, we define a new predicate $s\_3\_q^{ccf}$ using the rule

$$r6{:}s\_3\_q^{ccf}(X,U,W,Y) :- m\_q^{ccf}(X,Y), q1^{cf}(X,U), q2^{fc}(W,Y).$$

In the second step of this iteration, we unfold the definition of m_q^{ccf} (rules $mr31$ and $mr33$) into rule $r6$ to get

$$r61{:}s\_3\_q^{ccf}(X,U,W,Y) :- X > 10, Y > 10, q1^{cf}(X,U), q2^{fc}(W,Y).$$

$$r62{:}s\_3\_q^{ccf}(X,U,W,Y) :- s\_1\_p^{cf}(X1,V1), X > V1, Y > 10, q1^{cf}(X,U), q2^{fc}(W,Y).$$

In the third step of this iteration, we fold rule $r6$ into the body of $r3$ to get

$$r31{:}q^{ccf}(X,Y,Z) :- s\_3\_q^{ccf}(X,U,W,Y), q3^{bbf}(U,W,Z).$$

The final program obtained after these two iterations consists of the set of rules $\{r41, r43, r51, r53, r61, r62, r11, r21, r31\}$. This resulting program is range-restricted and can be seen to be the same as the program obtained by the GMT transformation.

7. COMBINING CONSTRAINT PROPAGATION WITH MAGIC REWRITING

Consider a CQL program P, with linear arithmetic constraints. We described two procedures, procedure `Gen_Prop_predicate-constraints` and procedure `Gen_Prop_QRP-constraints`, that preserved the core of P while propagating constraints that occur in the program P. (A combination of these procedures generated and propagated minimum QRP constraints.) The rewritten programs obtained using these two procedures preserve equivalence with respect to every possible query on the query predicate. An advantage of these techniques is that if the evaluation of the original program computed only ground facts, so do the evaluations of the rewritten programs. A shortcoming of these techniques for propagating constraints is that they are not able to take advantage of the pattern of constants in the actual query, known only at run time, or the actual set of facts in the database predicates.

Magic Templates [10] is a rewriting strategy that *is* able to take advantage of constants in the actual query, as well as the actual set of facts in the database predicates, thereby restricting the computation to facts that are potentially relevant to answering a given query. However, a shortcoming of this technique is that the bottom-up evaluation of the Magic Templates transformed program could compute constraint facts, even if the bottom-up evaluation of the original program computed only ground facts. There are two cases in which the Magic Templates transformation computes only ground facts:

- First, when we use the class of *bf* adornments, where an argument is bound (b) only if it is bound to a ground term; it is free (f) otherwise.
- Second, when we use the class of *bcf* adornments of Mumick et al. [9] for groundable programs, with grounding sips, in conjunction with their GMT algorithm.

An important question is how procedures `Gen_Prop_predicate-constraints` and `Gen_Prop_QRP-constraints` interact with the use of Magic Templates in these cases.

In the rest of this section, we describe this interaction in detail for the class of *bf* adornments. In Section 6.2, we considered the use of *bcf* adornments in the context of the GMT algorithm. In Section 7.7, we briefly consider how the interaction of Magic and procedures `Gen_Prop_predicate-constraints` and `Gen_Prop_QRP-constraints` can be explored further for the case of *bcf* adornments.

We use the following notation. Let P^{pred} be the program obtained from P using `Gen_Prop_predicate-constraints`, P^{qrp} be the program obtained from P using `Gen_Prop_QRP_constraints`, and P^{mg} be the program obtained from (adorned) program P using Magic Templates rewriting. (Thus, $P^{pred,qrp}$ is the program obtained by first applying procedure `Gen_Prop_predicate-constraints` to P and then applying procedure `Gen_Prop_QRP-constraints` on the resultant program P^{pred}, etc.)

7.1. *Problems Addressed*

Consider a *bf* adorned CQL program P, with linear arithmetic constraints, such that the bottom-up evaluation of P computes only ground facts. The program P can be successively optimized using a sequence of applications of procedure `Gen_Prop_predicate-constraints` (to generate and propagate minimum predicate constraints), procedure `Gen_Prop_QRP-constraints` (to generate and propagate QRP constraints), and the Magic Templates rewriting (to propagate binding information). An important question concerns their interaction.

Our main result is that given a *bf* adorned CQL program P, with linear arithmetic constraints, the rewritten program obtained by first applying procedure `Gen_Prop_predicate-constraints`, followed by applying procedure `Gen_Prop_QRP-constraints`, and finally adapting the Magic Templates rewriting, that is $P^{pred,qrp,mg}$, is optimal among *all* rewritten programs obtained from P by using a sequence of applications of the three rewritings, where the first two rewritings can be applied any number of times in the sequence, and the third rewriting can be applied exactly once.

We also show the following situations:

- Given a *bf* adorned CQL program P, with linear arithmetic constraints, procedure `Gen_Prop_QRP-constraints` and the Magic Templates rewriting are not confluent, i.e., the order in which these two rewritings are applied on a program affects the final resultant program. For some programs P, the program $P^{qrp,mg}$ computes fewer facts (for all EDBs and queries) than the program $P^{mg,qrp}$; for other programs, the program $P^{mg,qrp}$ computes fewer facts than $P^{qrp,mg}$. We also identify conditions when it is more advantageous to apply procedure `Gen_Prop_QRP-constraints` first.
- Given a *bf* adorned CQL program P, with linear arithmetic constraints, procedure `Constraint_rewrite` (which is a sequence of applications of procedures `Gen_Prop_predicate-constraints` and `Gen_Prop_QRP-constraints`) and the Magic Templates rewriting are not confluent either. However, as shown by our main result, the program $P^{pred,qrp,mg}$ (obtained by applying procedure `Constraint_rewrite` first and then applying the Magic Templates rewriting) always computes fewer facts than the program $P^{mg,pred,qrp}$.

The Magic Templates rewriting that we consider for our results is one in which all the constraint information present in a rule r in program P is also present in each magic rule generated from rule r. We refer to this as *constraint magic rewriting*, which is described next.

7.2. *Constraint Magic Rewriting*

Consider an adorned CQL program P. The Magic Templates rewriting of P to P^{mg} is said to be a *constraint magic rewriting* iff the following condition is satisfied. Let r be any rule in P of the form

$$r: p(\bar{X}) :- C_r, p_1(\overline{X_1}), \ldots, p_n(\overline{X_n}).,$$

where C_r is the conjunction of constraints in the body of rule r. Let mr_i be a magic

rule in P^{mg}, obtained from body literal $p_i(\overline{X_i})$ in rule r, of the form

$$mr_i : m\_p_i(\overline{X_i^1}) :- m\_p(\overline{X^1}), C_{mr_i}, \ldots,$$

where C_{mr_i} is the conjunction of constraints in the body of rule mr_i. For each such mr_i in P^{mg}, let $\overline{Y_i}$ be the variables appearing in the (head and body of the) rule. Then, for each rule r in P and each such mr_i in P^{mg}, it is the case that

$$\Pi_{\overline{Y_i}}(C_r) \equiv \Pi_{\overline{Y_i}}(C_{mr_i}).$$

The intuition is that all the constraints in rule r that are "relevant" to the magic rule mr_i should be present in the body of rule mr_i.

We make the following assumptions about the constraint magic rewriting used:

- The same sip strategy is chosen for rules that differ only in the constraints in the rule. This assumption is made for simplicity of exposition, and can be relaxed easily.
- The conjunction of constraints C_r in the body of rule r is in the tail of every sip arc for rule r. This guarantees that the Magic Templates rewriting is a constraint magic rewriting.

Proposition 7.1. Consider a bf adorned CQL program P, with linear arithmetic constraints, such that the bottom-up evaluation of P computes only ground facts. Consider a sip strategy S and let P^{mg} be the result of the constraint magic rewriting of P with sip strategy S. Then the bottom-up evaluation of P^{mg} computes only ground facts and is query equivalent to P for all input databases.

7.3. Combining `Gen_Prop_QRP-constraints` with Constraint Magic Rewriting

Each of the rewritings, procedure `Gen_Prop_QRP-constraints` and constraint magic rewriting, propagates information available on the head of a rule to predicates occurring in the body of the rule. Procedure `Gen_Prop_QRP-constraints` propagates constraint information, whereas constraint magic rewriting propagates information about the pattern of constants in the actual query and the facts in the actual database. Consequently, it is of interest to determine whether these two rewritings are confluent, i.e., does the order in which these two rewritings are applied on a program affect the final resultant program?

We first describe example programs to show that the two rewritings are *not* confluent. (The adornments are omitted for simplicity.)

Example 7.1. Consider the following program P:

$$r1: q(X,Y) :- a1(X,Y), X \leq 4.$$
$$r2: a1(X,Y) :- b1(X,Z), a2(Z,Y).$$
$$r3: a2(X,Y) :- b2(X,Y).$$
$$r4: a2(X,Y) :- b2(X,Z), a2(Z,Y).,$$

where q is the query predicate and q^{ff} is the query adornment. For this program P, it is preferable to apply procedure `Gen_Prop_QRP-constraints` followed by

the constraint magic rewriting, independent of the facts in the database. Example D.1 in Appendix D describes in more detail the various programs obtained by applying the rewritings in different orders.

The preceding example also illustrates that procedure `Constraint_rewrite` and constraint magic rewriting are *not* confluent either.

Example 7.2. Consider the following program P:

$r1{:}q(X,Y) :- a1(X,Y).$
$r2{:}a1(X,Y) :- b1(X,Z), X \leq 4, a2(Z,Y).$
$r3{:}a2(X,Y) :- b2(X,Y).$
$r4{:}a2(X,Y) :- b2(X,Z), a2(Z,Y).,$

where q is the query predicate and q^{bf} is the query adornment. For this program P, it is preferable to apply constraint magic rewriting followed by procedure `Gen_Prop_QRP-constraints`, independent of the facts in the database, and pattern of constants in the actual query. Example D.2 in Appendix D describes in more detail the various programs obtained by applying the rewritings in different orders.

Examples 7.1 and 7.2 show that, in general, no ordering of procedure `Gen_Prop_QRP-constraints` and constraint magic rewriting is always superior to the other. However, for a restricted class of CQL programs, we can show that applying procedure `Gen_Prop_QRP-constraints` followed by constraint magic rewriting is superior to applying constraint magic rewriting followed by applying procedure `Gen_Prop_QRP-constraints`. Theorem 7.2 identifies conditions on the form of such programs.

Theorem 7.2. Consider a bf adorned CQL program P, with sip strategy S such that the bottom-up evaluation of P computes only ground facts. Also, if r is a rule in P of the form

$$r{:}p(\bar{X}) :- C_r, p_1(\overline{X_1}), \ldots, p_n(\overline{X_n}).,$$

let C_r imply $PTOL(p_i(\overline{X_i}), C'_{p_i})$, $1 \leq i \leq n$, where C'_{p_i} is the minimum predicate constraint for predicate p_i. Then, for all databases D and pattern of constants in the actual query, the bottom-up evaluation of $P^{qrp,mg}$ (using sip strategy S) on database D computes a subset of the set of facts computed by the bottom-up evaluation of $P^{mg,qrp}$ (using sip strategy S) on database D.

PROOF. Consider a CQL program P satisfying the conditions of the theorem. Consider any rule r in P:

$$r{:}p(\bar{X}) :- C_r, p_1(\overline{X_1}), \ldots, p_n(\overline{X_n}).$$

In the magic program P^{mg}, the modified rule and the magic rules generated from rule r in P are of the form

$$r{:}p(\bar{X}) :- C_r, m\_p(\overline{X^1}), p_1(\overline{X_1}), \ldots, p_n(\overline{X_n}).$$
$$mr_1{:}m\_p_1(\overline{X_1^1}) :- C_r, m\_p(\overline{X^1}).$$
$$\ldots$$
$$mr_n{:}m\_p_n(\overline{X_n^1}) :- C_r, m\_p(\overline{X^1}), \ldots .$$

Again, consider rule r in P and the propagation of QRP constraints into the body of rule r using procedure `Gen_Prop_QRP-constraints`. Each of the resulting rules[10] in P^{qrp} is of the form

$$r{:}p(\bar{X}){:}-C_p, C_r, p_1(\overline{X_1}), \ldots, p_n(\overline{X_n}).,$$

where C_p is a disjunct in the QRP constraint propagated for p. Because the rules in P^{qrp} have at least as many constraints as the corresponding rule r in P, constraint magic rewriting guarantees that each rule in $P^{qrp,mg}$ obtained from rule r in P^{qrp} has at least as many constraints as the corresponding rules in P^{mg}.

Hence, to prove the theorem we only need to prove that the QRP constraints generated and propagated by procedure `Gen_Prop_QRP-constraints` for each (magic and nonmagic) derived predicate p in P^{mg} are implied by the conjunction of constraints present in the body of each rule defining p in $P^{qrp,mg}$.

First consider a nonmagic derived predicate p in P^{mg} and $P^{qrp,mg}$. The constraints present in the body of each rule defining p in $P^{qrp,mg}$ include, in addition to the constraints present in the corresponding rule in P^{mg}, a disjunct from the QRP constraint generated for p in P (and propagated in P^{qrp}).

Claim 1. If $C1_p$ is the QRP constraint generated for predicate p in P, and $C2_p$ is the QRP constraint generated for predicate p in P^{mg}, then $C1_p \supset C2_p$.

PROOF OF CLAIM 1. The QRP constraint obtained for p in P was based on occurrences of p literals in the body of rules in P and corresponding literal constraints. In P^{mg}, each of these occurrences is present (in modified original rules). In addition, there are some *more* occurrences of p literals in the bodies of magic rules. Using induction, it can be shown that at the end of each iteration i of procedure `Gen_QRP-constraints`, the "approximate" QRP constraint $C1_p^i$ (the disjunction of $C1_p^{i-1}$ and the literal constraints on occurrences of p in the ith iteration) implies the "approximate" QRP constraint $C2_p^i$. The claim follows from the monotonic nature of the nonrecursive inference in the generation of QRP constraints. □

Next, consider a magic predicate m_p in P^{mg} and $P^{qrp,mg}$.

Claim 2. If $C1_{m\_p}$ is the conjunction of constraints in the body of a rule defining $m\__p$ in $P^{qrp,mg}$ and $C2_{m\_p}$ is the QRP constraint generated for predicate m_p in P^{mg}, then $C1_{m\_p} \supset C2_{m\_p}$.

PROOF OF CLAIM 2. (Sketch). The QRP constraint generated for predicate m_p in P^{mg} is obtained from rules containing body occurrences of m_p. These are the (modified original) rules defining p in P^{mg} and the magic rules obtained from the rules defining p in P. Using simultaneous induction on the iterations of procedure `Gen_QRP-constraints` and procedure `Gen_predicate-constraints`, it can be shown that $C2_{m\_p}$, the QRP constraint generated for predicate m_p in P^{mg}, is implied by the predicate constraint C_p' for predicate p in P. By hypothesis, the conjunction of constraints in the body of each rule containing a body occurrence of p implies the *PTOL* of C_p' on this literal. Magic rules defining m_p in $P^{qrp,mg}$ are obtained from such body occurrences of p. The claim follows from the property of constraint magic rewriting (the relationship between the constraints present in a

[10] There could be many such rules if the QRP constraint propagated was a nontrivial disjunction.

magic rule and the rule from which it was generated) in obtaining $P^{qrp,mg}$ from P^{qrp}.

This concludes the proof of the theorem. □

7.4. *Combining* `Gen_Prop_predicate-constraints` *and* `Gen_Prop_QRP-constraints`

Consider a CQL program P, with linear arithmetic constraints, such that the bottom-up evaluation of P computes only ground facts. In this section, we show several results about the rewritten program obtained by combining procedures `Gen_Prop_predicate-constraints` and `Gen_Prop_QRP-constraints` on program P.

Theorem 7.3. Consider a CQL program P, with linear arithmetic constraints, such that the bottom-up evaluation of P computes only ground facts. Then the bottom-up evaluation of $P^{pred,qrp}$ computes a subset of the facts computed by the bottom-up evaluation of $P^{qrp,pred}$.

PROOF. Consider a CQL program P satisfying the conditions of the theorem. From Theorem 4.8, it follows that the evaluation of $P^{pred,qrp}$ computes a subset of the facts computed by the evaluation of P^{qrp}. From Theorem 4.6, it follows that the evaluation of P^{qrp} computes the same set of facts as the evaluation of $P^{qrp,pred}$. Combining these two results, we have a proof of the theorem. □

The following result indicates that consecutive applications of procedure `Gen_Prop_predicate-constraints` on a program are redundant.

Theorem 7.4. Consider a CQL program P, with linear arithmetic constraints. Then for each predicate p, the minimum predicate constraint $C1_p$ on predicate p in P^{pred} is equivalent to C_p, the minimum predicate constraint on predicate p in P.

If the bottom-up evaluation of P computes only ground facts, the bottom-up evaluation of $P^{pred,pred}$ computes the same set of facts for each program predicate as the bottom-up evaluation of P^{pred}.

The first half of the result follows from the definition of minimum predicate constraints. The second half of the result is a corollary of Theorem 4.6.

The following result indicates that consecutive applications of procedure `Gen_Prop_QRP-constraints` on a program are redundant.

Theorem 7.5. Consider a CQL program P, with linear arithmetic constraints. Then for each predicate p, the QRP constraint $C1_p$ generated by procedure `Gen_QRP-constraints` *on predicate p in P^{qrp} is equivalent to C_p, the QRP constraint generated by procedure* `Gen_QRP-constraints` *on predicate p in P.*

If the bottom-up evaluation of P computes only ground facts, the bottom-up evaluation of $P^{qrp,qrp}$ computes the same set of facts for each program predicate as the bottom-up evaluation of P^{qrp}.

The result can be shown by induction of the iterations of procedure `Gen_QRP-constraints` on P and P^{qrp}.

From Theorems 7.4 and 7.5, it follows that in rewriting a program using `Gen_Prop_predicate-constraints` and `Gen_Prop_QRP-constraints`,

one only needs to alternate between the two rewritings; consecutive applications of the same rewriting are redundant.

From Theorem 4.8, we know that $P^{pred,qrp}$ is the rewritten program obtained by generating and propagating the minimum QRP constraints for predicates in P. The next result shows that more than one alteration of procedures `Gen_Prop_predicate-constraints` and `Gen_Prop_QRP-constraints` is redundant.

Theorem 7.6. Consider a CQL program P, with linear arithmetic constraints. Then:

- *The minimum predicate constraint $C1_p$ on predicate p in $P^{pred,qrp}$ is equivalent to C_p, the minimum QRP constraint on predicate p in P.*
- *The minimum QRP constraint $C2_p$ on predicate p in $P^{pred,qrp}$ is equivalent to C_p, the minimum QRP constraint on predicate p in P.*

A corollary to the second part of Theorem 7.6 follows.

Corollary 7.7. Consider a CQL program P, with linear arithmetic constraints. If the bottom-up evaluation of P computes only ground facts, P^{S1}, where S1 is the sequence {pred, qrp, pred, qrp}, computes the same set of facts for each program predicate as $P^{pred,qrp}$.

7.5. *Adding Constraint Magic Rewriting*

Constraint magic rewriting requires that the predicates be adorned first. Because adorning a program can make different occurrences of the same predicate p have different adornments and thus be treated as different predicates, it helps to adorn the program prior to applying any of procedures `Gen_Prop_predicate-constraints` and `Gen_Prop_QRP-constraints`.

Consider a *bf* adorned CQL program P, with linear arithmetic constraints. In this section, we discuss properties about a sequence of applications of procedures `Gen_Prop_predicate-constraints`, `Gen_Prop_QRP-constraints`, and constraint magic rewriting on P. Example 7.1 shows that constraint magic rewriting and the sequence of rewritings, procedures `Gen_Prop_predicate-constraints` and `Gen_Prop_QRP-constraints` are not confluent; that is, $P^{pred,qrp,mg}$ does not compute the same set of facts as $P^{mg,pred,qrp}$. In the example, $P^{pred,qrp,mg}$ computed fewer facts than $P^{mg,pred,qrp}$ for all input databases. We now show that this is true, in general.

Theorem 7.8. Consider a bf adorned CQL program P, with linear arithmetic constraints, such that the bottom-up evaluation of P computes only ground facts. Then for all databases D and pattern of constants in the actual query, the bottom-up evaluation of $P^{pred,qrp,mg}$ on database D computes a subset of the set of facts computed by the bottom-up evaluation of $P^{mg,pred,qrp}$ on database D.

PROOF. Consider a CQL program P satisfying the conditions of the theorem. Using arguments similar to the proof of Theorem 7.2, it can be seen that each rule in $P^{pred,qrp,mg}$ has at least as many constraints as the corresponding rules in P^{mg}.

Hence, to prove the theorem we only need prove that the minimum QRP constraints generated and propagated for each (magic and nonmagic) derived

predicate p in P^{mg} are implied by the conjunction of constraints present in the body of each rule defining p in $P^{pred,qrp,mg}$. First, we prove certain results about the application of procedure `Gen_Prop_predicate-constraints` on P^{mg}.

Claim 1. The minimum predicate constraints generated for each nonmagic predicate p and the corresponding magic predicate m_p in P^{mg} by procedure `Gen_predicate-constraints` are implied by the minimum QRP constraints generated for predicate p in P.

PROOF OF CLAIM 1. (Sketch). The claim is proved by induction on the SCC structure of P^{mg}.

For the base case, consider the magic predicate m_q corresponding to the query predicate q in P^{mg} and the database predicates in P^{mg}. The minimum predicate constraint on the magic predicate m_q is given to be **true**, because every query is possible. Because the minimum predicate constraints on the database predicates in P are the same as the minimum predicate constraints on the corresponding predicates, in P^{mg}, the minimum QRP constraints on the database predicates in P trivially imply the minimum predicate constraints on the database predicates in P^{mg}.

Now consider the induction step. Consider SCC S_i of P^{mg} and predicate m_p in S_i. The minimum predicate constraint obtained for m_p in P^{mg} is based on the predicate occurrences in the body of the rules defining m_p and the minimum predicate constraints for those predicates in P^{mg}. For each rule mr_i in P^{mg} defining m_p, consider the rule r in P from which it was generated. The body of rule r in P contains occurrences of all the body predicates and constraints occurring in the body of mr_i. In the process of computing the minimum QRP constraint for p in P, procedure `Gen_Prop_predicate-constraints` propagated the minimum predicate constraint associated with each literal in the body of r in P. Theorem 7.6 guarantees that the minimum QRP constraint for p in P would not be different had the minimum QRP constraint been associated with each literal in the body of r in P. By the induction hypothesis, this minimum QRP constraint for a predicate p_1 in the body of r implies the minimum predicate constraint for the corresponding predicate p_1 in the body of mr_i, if p_1 was defined in a lower SCC of P^{mg}. If p_1 in the body of mr_i is not defined in a lower SCC of P^{mg}, it has to be defined in S_i. The proof now requires an additional induction on the iterations performed by procedure `Gen_predicate-constraints` on P^{mg}.

From this it follows that the minimum predicate constraint on m_p in P^{mg} is implied by the minimum QRP constraint on p in P.

Consider now predicate p defined in S_i of P^{mg}. Again, the minimum predicate constraint for p depends on the minimum predicate constraints associated with predicates in the bodies of rules defining p. The rules defining p in P^{mg} are similar to rules defining p in P. The only additional literal is an occurrence of m_p. Again, an application of Theorem 7.6 and the form of the minimum predicate constraints for m_p ensures that the minimum predicate constraint on p in P^{mg} is implied by the minimum QRP constraint on p in P. This concludes the induction step and the proof of the claim. □

Claim 2. If $C1_p$ is the minimum QRP constraint generated for predicate p in P and $C2_p$ is the minimum QRP constraint generated for predicate p in P^{mg}, then $C1_p \supset C2_p$.

PROOF OF CLAIM 2. This is very similar to the proof of Claim 1 in the proof of Theorem 7.2, except that it additionally requires an application of Theorem 7.6 along with the foregoing Claim 1. □

Claim 3. If $C1_{m\_p}$ is the conjunction of constraints in the body of a rule defining m_p in $P^{pred,qrp,mg}$ and $C2_{m\_p}$ is the minimum QRP constraint generated for predicate m_p in P^{mg}, then $C1_{m\_p} \supset C2_{m\_p}$.

PROOF OF CLAIM 3. This is very similar to the proof of Claim 2 in the proof of Theorem 7.2, except that it additionally requires an application of Theorem 7.6 along with the foregoing Claim 1. By combining Claims 2 and 3, we have a proof of the theorem. □

The following theorem shows that generating and propagating predicate constraints on $P^{pred,qrp}$ prior to constraint magic rewriting is redundant.

Theorem 7.9. Consider a bf adorned CQL program P, with linear arithmetic constraints, such that the bottom-up evaluation of P computes only ground facts. Then the bottom-up evaluation of P^{S1}, where S1 is the sequence {pred, qrp, pred, mg} computes the same set of facts for each predicate as the bottom-up evaluation of P^{S2}, where S2 is the sequence {pred, qrp, mg}.

PROOF. Consider a CQL program P satisfying the conditions of the theorem.

Claim 1. Consider a rule r in $P^{pred,qrp}$ of the form

$$r: p(\overline{X}) :- C_r, p_1(\overline{X_1}), \ldots, p_n(\overline{X_n}).,$$

where C_r is the conjunction of constraints in the body of the rule. Let C_{p_i} be the minimum QRP constraint on predicate p_i in P. Then,

$$LTOP\big(p_i(\overline{X_i}), \Pi_{\overline{X_i}}(C_r)\big) \supset C_{p_i}.$$

PROOF OF CLAIM 1. The proof is a direct consequence of the fact that the conjunction of constraints in the body of each rule in $P^{pred,qrp}$ is stronger than the conjunction of constraints in the body of the corresponding rule in P. □

Claim 2. Let $p_i(\overline{X_i}), \ldots, p_i(\overline{X_m})$ be all the body occurrences of p_i in $P^{pred,qrp}$. Let C_{p_i} be the minimum QRP constraint on predicate p_i in P. Then

$$\bigvee_{j=1}^{m} LTOP\big(p_i(\overline{X_j}), \Pi_{\overline{X_j}}(C_{r,j})\big) \equiv C_{p_i},$$

where $C_{r,j}$ is the conjunction of constraints in the body of the rule containing the occurrence $p_i(\overline{X_j})$.

PROOF OF CLAIM 2. The conjunction of constraints $LTOP(p_i(\overline{X_j}), \Pi_{\overline{X_j}}(C_{r,j}))$ restricts the set of p_i facts that can be used in literal $p_i(\overline{X_j})$. Because C_{p_i} is the *minimum* QRP constraint on predicate p_i, each ground p_i fact that satisfies C_{p_i} must satisfy at least one of the *LTOP*s of the various p_i literals, else that fact would be a witness for the nonminimality of C_{p_i}. Consequently, $C_{p_i} \supset$ the disjunc-

tion of the *LTOP*s. From Claim 1, we have that the disjunction of the *LTOP*s $\supset C_{p_i}$. Combining the two, we have a proof of the claim.

From Theorem 7.6, we know that the minimum predicate constraint $C1_p$ on predicate p in $P^{pred,qrp}$ is equivalent to C_p, the minimum QRP constraint on predicate p in P. From the preceding two claims, it follows that propagating $C1_p$ (using `Gen_Prop_predicate-constraints`) would not change the constraints associated with any rule. Consequently, applying constraint magic rewriting on $P^{pred,qrp}$ and $P^{pred,qrp,pred}$ would result in equivalent corresponding (nonmagic and magic) rules. This proves the desired result. □

7.6. *An Optimal Sequence of Transformations*

We now show that $P^{pred,qrp,mg}$ is optimal among a class of transformation sequences on program P.

Theorem 7.10. Consider a bf adorned CQL program P, with linear arithmetic constraints, such that the bottom-up evaluation of P computes only ground facts. Consider the class of all programs obtained from P using a sequence of applications of procedure `Gen_Prop_predicate-constraints`, *procedure* `Gen_Prop_QRP-constraints`, *and constraint magic rewriting, such that constraint magic rewriting is applied only once. Among all such programs,* $P^{pred,qrp,mg}$ *is optimal in that it computes a subset of the facts computed by any other program from this class, for all input databases.*

PROOF. Consider a CQL program P satisfying the conditions of the theorem, and consider the class of programs obtained from P using a sequence of applications of procedure `Gen_Prop_predicate-constraints`, procedure `Gen_Prop_QRP-constraints`, and constraint magic rewriting, such that constraint magic rewriting is applied only once.

Such programs can be denoted by P^{S1}, where $S1$ is the sequence of transformations

$$\{pred^{i_1}, qrp^{j_1}, \ldots, pred^{i_k}, qrp^{j_k}, mg, pred^{i_{k+1}}, qrp^{j_{k+1}}, \ldots, pred^{i_n}, qrp^{j_n}\}.$$

From Theorems 7.4, 7.5, and 7.6, it follows that any such program computes the same set of facts for each program predicate as the program P^{S2}, where $S2$ is the sequence of transformations {*pred*, *qrp*, *pred*, *mg*, *pred*, *qrp*}. From Theorem 7.8, it follows that the program P^{S3}, where $S3$ is the sequence {*pred*, *qrp*, *pred*, *pred*, *qrp*, *mg*} computes a subset of the set of facts computed by P^{S2}, for all input databases. Using another application of Theorems 7.4, 7.6, and 7.9, we have that P^{S3} is equivalent to $P^{pred,\ qrp,\ mg}$ for all input databases. This gives us the desired optimality result. □

The importance of Theorem 7.10 is in prescribing an optimal order in which to apply `Gen_Prop_predicate-constraints`, `Gen_Prop_QRP-constraints`, and the constraint magic rewriting on *bf* adorned program P, if we desire a rewritten program P', such that the bottom-up evaluation of P':

- utilizes constraint information present in the program P,
- utilizes the actual facts present in the database and the pattern of constants in the actual query, and
- computes only ground facts.

To summarize, the optimal order for rewriting *bf* adorned CQL program P, with linear arithmetic constraints, is as follows:

1. Apply procedure `Gen_Prop_predicate-constraints` on P to generate and propagate minimum predicate constraints. Let the resultant program be P^{pred}.
2. Apply procedure `Gen_Prop_QRP-constraints` on P^{pred} to generate and propagate minimum QRP constraints. Let the resultant program be $P^{pred,qrp}$.
3. Apply constraint magic transformation on $P^{pred,qrp}$ to obtain $P^{pred,qrp,mg}$.

The program $P^{pred,qrp,mg}$ is the resultant optimized program.

7.7. *Other Classes of Adornments*

A natural question to ask is how do procedures `Gen_Prop_predicate-constraints`, `Gen_Prop_QRP-constraints`, and constraint magic rewriting interact when given a program P adorned using a class of adornments different from *bf*. One such possibility is the class of *bcf* adornments of Mumick et al. [9], where the '*c*' adornment denotes an argument that is independently constrained.

Given a *bcf* adorned CQL program P, with linear arithmetic constraints, such that the bottom-up evaluation of P computes only ground facts, it can be seen easily that the rewritten program obtained by applying either of procedures `Gen_Prop_predicate-constraints` or `Gen_Prop_QRP-constraints` also computes only ground facts. Further, each of these transformations also preserves the semantics of the '*c*' adornment. Intuitively, this is because these transformations only add constraints to rule bodies without altering the core of the program.

The main problem is that for the class of *bcf* adornments, constraint magic rewriting alone does not guarantee that the evaluation of the rewritten program P^{mg} computes only ground facts, even if the evaluation of the original program P computed only ground facts. However, for groundable programs [9], one could replace the constraint magic rewriting described in Section 7.2 by (a suitably enhanced version of) the GMT algorithm to obtain a magic rewriting that computes only ground facts.

We conjecture that our results for the class of *bf* adornments can be extended to the class of groundable programs with *bcf* adornments as well. The intuition behind this conjecture is that each of the three transformations for the class of groundable programs with *bcf* adornments preserves the semantics of the '*c*' adornment. Further, none of the proofs of the various results relied on the actual adornment class used by the program.

8. CONCLUSIONS AND FUTURE DIRECTIONS

We formally defined the problem of propagating constraints occurring in a program without altering the syntactic program structure, using the notion of minimum query-relevant predicate constraints. If these constraints are propagated into a program, the rewritten program is query-equivalent to the original program (on

all input EDBs) and computes only facts that are constraint-relevant to the program on an input EDB. We showed that the problem of determining whether any representation for the minimum QRP constraints for program predicates is finite is undecidable, in general, for linear arithmetic constraints. We presented two procedures: (1) for generating and propagating minimum predicate constraints based on the definition of predicates and (2) for generating and propagating query-relevant predicate constraints based on the uses of predicates, and showed that a combination of these procedures generates and propagates minimum QRP constraints if it terminates. It is the generation aspect that is nonterminating; once we have a finite minimum QRP constraint, the propagation uses fold/unfold transformations and always terminates. We also identified a class of programs for which our procedure terminates.

We described a uniform framework—namely, a combination of Magic Templates and (possibly simpler versions of) our procedures for generating and propagating minimum predicate constraints and QRP constraints—for the results of this paper and for related work in the literature for propagating constraint selections. By considering semantic manipulation of constraints, the techniques presented here significantly extend earlier work.

We studied the interaction of our procedures (for generating and propagating minimum predicate constraints and QRP constraints) with the Magic Templates transformation for a class of programs where the Magic Templates evaluation computes only ground facts if the original program computed only ground facts. We showed the optimality (among a class of transformation sequences where the Magic Templates rewriting is applied exactly once) of a transformation sequence that applies the Magic Templates transformation *after* generating and propagating minimum QRP constraints.

There are several directions in which this work can be extended. One important direction is to identify classes of programs for which there is a terminating procedure to compute minimum predicate constraints and minimum QRP constraints. A promising candidate is the class of programs called "strongly unique" programs by Brodsky and Sagiv [3].

By first propagating predicate constraints, it may be possible for a Magic Templates evaluation to terminate, whereas the evaluation may not have terminated otherwise. Another interesting problem is to study this interaction between our procedure for computing minimum predicate constraints and the Magic Templates transformation when the evaluation computes constraint facts.

APPENDIX A FOLD/UNFOLD TRANSFORMATIONS

In this section, we formally define the *fold*, *unfold*, and *definition* steps for programs in a constraint query language, restricted to the transformations required for our purposes.

The set P_i, $i \geq 0$, is the set of rules in the program obtained after applying i definition, fold, or unfold steps. The set N_i, $i \geq 0$, is the set of rules defining new predicates after applying i definition, fold, or unfold steps. At any step, a definition, fold, or unfold step may be applied to produce P_i and N_i from P_{i-1} and N_{i-1}, for $i > 0$. The set P_0 is the set of rules in the initial program P, and the set of rules defining new predicates, N_0, is initially $\varnothing$.

Definition Step

1. Let $r_1, \ldots, r_m$ be m rules of the form

$$r_1: p'(\bar{X}) :- C_1(\bar{X}), p(\bar{X}).$$

$$r_m: p'(\bar{X}) :- C_m(\bar{X}), p(\bar{X}).,$$

where the variables in $\bar{X}$ are distinct variables, each $C_i(\bar{X})$, $1 \leq i \leq m$, is a conjunction of constraints, p' is a predicate not appearing in P_{i-1}, N_{i-1}, and p is a predicate appearing in P_0.

2. The updated set of program rules is given by $P_i = P_{i-1} \cup \{r_1, \ldots, r_m\}$. The updated set of rules defining new predicates is given by $N_i = N_{i-1} \cup \{r_1, \ldots, r_m\}$.

Note that because the variables occurring in the head of the rule are all and only the variables occurring in the body of the rule, the problems described in [5] do not apply.

Unfolding Step

1. Let r be a rule in P_{i-1}, $p(\bar{X})$ a body literal occurring in r, and let $r_1, \ldots, r_n$ be all the rules in P_{i-1} whose head literals are unifiable with $p(\bar{X})$.
2. Let r_j', $1 \leq j \leq n$, be the result of resolving r with r_j upon $p(\bar{X})$.
3. The updated set of program rules is given by $P_i = (P_{i-1} - \{r\}) \cup \{r_1', \ldots, r_n'\}$. The set of rules defining new predicates remains unchanged, i.e., $N_i = N_{i-1}$.

Folding Step

1. Let r be a rule in P_{i-1} of the form

$$r: p_0(\overline{X_0}) :- C_r(\bar{Y}), C_1(\overline{X_1}), p_1(\overline{X_1}), \ldots, C_n(\overline{X_n}), p_n(\overline{X_n}).,$$

where each $C_i(\overline{X_i})$, $1 \leq i \leq n$, is a conjunction of constraints on the variables in $p_i(\overline{X_i})$, and $C_r(\bar{Y})$ is also a conjunction of constraints. Let $r1$ be a rule in N_{i-1} of the form

$$r1: p'(\bar{X}) :- C(\bar{X}), p_i(\bar{X}).,$$

where $\bar{X}$ is a tuple of distinct variables and $C(\bar{X})$ is a conjunction of constraints.

2. Let there be a substitution θ and a body literal $p_i(\overline{X_i})$ in r such that $p_i(\overline{X_i}) = p_i(\bar{X})\theta$ and $C_i(\overline{X_i}) \supset C(\bar{X})\theta$.
3. Let r' be a rule obtained from r by deleting the literal $p_i(\overline{X_i})$ from the body of the rule and adding $p(\bar{X})\theta$ to the body.
4. The folding step is described by $P_i = (P_{i-1}\{r\}) \cup \{r'\}$; $N_i = N_{i-1}$.

Note that we do not mark rules as "foldable" or not, as is done in [1]. The procedure that uses these steps to rewrite a program in a constraint query language ensures that no undesirable folds (like a rule being folded by itself) occur.

APPENDIX B MAGIC TEMPLATES TRANSFORMATIONS

Magic Templates transformations are used to imitate top-down computations using bottom-up computation. The major advantage they provide is that they allow a bottom-up computation to be specialized with respect to the query, thus improving the efficiency of answering queries. A brief description of the Magic Templates transformation is presented here, and the reader is referred to [10] and [16] for further details.

First, we define the concepts of sideways information passing strategies (sips). An argument of a literal in a rule is considered to be *bound* if that argument is bound (not necessarily to a constant) in the evaluation under consideration. Intuitively, for a rule of a program, a sip represents a decision about the order in which the literals of the rule are to be evaluated when a given set of head arguments is known to be bound; different sips can be chosen for each head binding pattern.

Definition B.1 (Sips). Let R be a rule and let p_h be the head literal restricted to the set of bound arguments. Let $Lits(R)$ be the set of literals in the body of R. A *sideways information passing strategy* (or sips) for rule R is a labeled graph that satisfies the following conditions:

1. Each node is either a subset or a member of $Lits(R) \cup \{p_h\}$.
2. Each arc is of the form $N \rightarrow_{\chi} q$, where N is a subset of $Lits(R) \cup \{p_h\}$, q is a member of $Lits(R)$, and χ is a set of variables, such that each variable of χ appears in q.
3. There exists a partial ordering of the literals in $Lits(R) \cup \{p_h\}$ such that (a) p_h is first, (b) for each arc, all the literals in its tail precede the literal at its head, and (c) the literals that do not appear in the sips follow all others.

A sips for a program consists of a sips for each rule in the program.

A binding pattern, or *adornment*, for an n-ary predicate p can be represented as a string a of length n on the alphabet $\{b, f\}$, where b stands for *bound* and f stands for *free*. At compile time, we can compute the binding patterns, also called adornments, for predicates that arise during the evaluation of a given query, for a given choice of sips.

Definition B.2 (Adorned program). Let P be a program, S be any sip strategy for P, and Q be a query of P. The adorned version of the program AP is obtained as follows.

1. For each derived predicate p, for each rule that has p as its head predicate, and for each adornment a for p, we construct a new adorned version of the rule. The predicate p in the head is replaced by the adorned predicate p^a. A sip that matches p^a is chosen from S. Next, each derived predicate in the body of the rule is replaced by an adorned version, obtained as follows.[11] We replace the $p_i(\ )$ by $p_i^{a_i}(\ )$ where an argument position in a_i is marked bound when:
   - the argument in that position is a constant, or

[11] For simplicity, we assume that the sip strategy S has at most one arc entering a given literal. The reader is referred to [10] for the general case.

- the variable in that position appears in the label of the sip arc entering the literal.

The arguments of the literal in the new rule remain unchanged. We have thus replaced the original predicates and rules by a collection of adorned predicates and rules.

2. We replace the query by an adorned version. If the query predicate is q, the actual query determines bindings for q, and we replace q by the appropriate adorned version.
3. Finally, we eliminate adorned predicates and rules defining these predicates that are not reachable from the query.

Intuitively, an adorned literal p^a corresponds to an evaluation of the predicate p with some arguments bound and the other arguments free, as indicated by the adornment.

Definition B.3 (Magic Templates transformation). Let P be a program, S be any sip strategy for P, and Q be a query of P. The *Magic Templates transformation* results in a new program MP obtained as follows. Initially, MP is empty.

1. First, create an adorned version AP of P.
2. Create a new predicate m_p for each adorned predicate p in AP, where the arity of m_p is the number of bound arguments in the adornment for p.
3. For each rule in AP, add the *modified version* of the rule to MP. If a rule has head $p(\bar{s})$, the modified version of this rule is obtained by adding the literal $m\_p(\bar{s}^b)$ to the body, where $\bar{s}^b$ is the set of bound argument positions of $\bar{s}$.
4. For each rule R in AP with head $p(\bar{s})$ and for each literal $q(\bar{t})$, add a *magic rule* to MP. The head is $m\_q(\bar{t}^b)$. The body contains the literal $m\_p(\bar{s}^b)$ if the tail contains the special literal p_h, and all the literals in the body of R that are in the tail of the sip arc in S with head $q(\bar{t})$.
5. Create a *seed* fact $m\_q(\bar{c}^b)$ from the query Q.

The intuition behind the magic sets rewriting is to compute a set of auxiliary (magic) predicates that contain the goals. The rules in the program are then modified by attaching additional literals that act as filters and prevent the rule from generating irrelevant facts.

APPENDIX C C ALGORITHMS

The following procedure generates and propagates QRP constraints, though not the minimum possible, for each derived predicate of a program P, if it terminates:

```
Gen_Prop_QRP-constraints (P)
{
  let p_1,...,p_m be the predicates defined in the program P, and
    let q be the query predicate.
  Gen_QRP-constraints (P).
  let C_p1,...,C_pm be the QRP-constraints obtained.
  let p'_1,...,p'_m be new predicates, not occurring in the program.
  for j = 1 to m do
    let C_pj have k_j disjuncts.
```

perform a *definition* step creating k_j rules with head $p_j'(\bar{X})$, and the sole body literal $p_j(\bar{X})$.
each rule has $PTOL(p_j(\bar{X}), C)$ as the conjunction of constraints in the body, where C is one of the k_j disjuncts.
end for
for $j = 1$ to m do
unfold the definition of p_j in P into each of the rules defining p_j'.
end for
for $j = 1$ to m do
fold the original definition of p_j' into rules in P containing body occurrences of p_j.
end for
the resultant program P' has all the QRP-constraints propagated.
}

```
Gen_QRP-constraints (P)
{
  let p_1,...,p_m be the predicates defined in the program P.
  let q (one of the p_i 's) be the query predicate.
  for i = 1 to m do
    C1_{p_i} = false.
  end for
  C1_q = true.
  repeat
    assuming C1_{p_i} as a QRP-constraint for each p_i, obtain literal
      constraints C_{p_k(X_i)} for each literal in each rule in P.
    for i = 1 to m do
      C2_{p_i} = ∨ each LTOP(p_i(X_l), C_{p_i(X_l)}) for p_i(X_l) in a rule in P.
    end for
    for i = 1 to m do
      if (C2_{p_i} ⊃ C1_{p_i}) then
        'mark' p_i.
      else
        'unmark' p_i.
        C1_{p_i} = C1_{p_i} ∨ C2_{p_i}.
      end if
    end for
  until (all predicates are 'marked')
  C1_{p_i} is the QRP-constraint obtained for each p_i.
}
```

The following procedure generates minimum predicate constraints for each derived predicate of a program P:

```
Gen_Prop_predicate-constraints (P)
{
  let p_1,...,p_m be the predicates defined in the program P.
```

let $b_1,\ldots,b_n$ be the database predicates.
let $C1_{b_1},\ldots,C1_{b_n}$ be the minimum predicate constraints fo database predicates.
/* These predicate constraints are part of the input. */
Gen_predicate-constraints$(P, C1_{b_1},\ldots,C1_{b_n})$.
let $C1_{p_1},\ldots,C1_{p_m}$ be the predicate constraints obtained.
for each rule in P of the form: $r_i: p(\bar{X}) :- C_{r_i}, p_{i1}(\overline{X_{i1}}),\ldots,p_{ik}(\overline{X_{ik}})$ do
let each $C1_{p_{ij}}, 1 \le j \le k$ have c_{ij} disjuncts.
create $c_{i1} * \cdots * c_{ik}$ new rules of the form:
$r_{i,h}: p(\bar{X}) :- C_{r_i}, C3'_{p_{i1}}, p_{i1}(\overline{X_{i1}}),\ldots,C3'_{p_{ik}}, p_{ik}(\overline{X_{ik}})$
where each $C3'_{p_{ij}}$ is a disjunct of $PTOL(p_{ij}(\overline{X_{ij}}), C1_{p_{ij}})$
end for
the resultant program is composed of the new set of rules.
}

Gen_predicate-constraints$(P, C1_{b_1},\ldots,C1_{b_n})$
{
let $p_1,\ldots,p_m$ be the predicates defined in the program P.
let $b_1,\ldots,b_n$ be tne database predicates, and
$C1_{b_i}, 1 \le i \le n$ the corresponding minimum predicate constraints.
let $\mathcal{R}$ be the rules in P defining p_i, $1 \le i \le m$.
for $i = 1$ to m do
$C1_{p_i} =$ **false**.
end for
repeat
Single_step$(\mathcal{R}, C1_{p_1},\ldots,C1_{p_m}, C1_{b_1},\ldots,C1_{b_n})$.
for $i = 1$ to m do
if $(C2_{p_i} \supset C1_{p_i})$ then
'mark' p_i.
else
'unmark' p_i.
$C1_{p_i} = C1_{p_i} \vee C2_{p_i}$.
end if
end for
until (all predicates are 'marked')
$C1_{p_i}$ is the minimum predicate constraint obtained for each p_i.
}

Single_step$(\mathcal{R}, C1_{p_1},\ldots,C1_{p_m}, C1_{b_1},\ldots,C1_{b_n})$
{
for $i = 1$ to m do
$C2_{p_i} =$ **false**.
end for
let the rules in $\mathcal{R}$ be $r_1,\ldots,r_k$.
for $i = 1$ to k do
let rule r_i be of the form: $r_i: p_j(\overline{X_j}) :- C_i(\bar{X}), p_{i1}(\overline{X_{i1}}),\ldots,p_{is}(\overline{X_{is}})$.
$C2_{p_j} = C2_{p_j} \vee LTOP(p_j(\overline{X_j}), \Pi_{\overline{X_j}}(C_i(\bar{X}) \& \wedge_{h=1}^{s} PTOL(p_{ih}(\overline{X_{ih}}), C1'_{p_{ih}})))$

```
    for each choice C1'_{p_ih} of a disjunct from C1_{p_ih}.
    /* The disjunction of the LTOPs is the inferred head
    constraint for rule r_i. */
  end for
  return C2_{p_1},...,C2_{p_m}.
```

The following procedure generates and propagates minimum QRP constraints for each derived predicate of a program P, if it terminates:

```
Constraint_rewrite (P)
{
  let q be the query predicate.
  define a new predicate q_1 with the same arity as q, and let
  the only rule defining q_1 be
    q_1(X_1):-q(X_1).
    where X_1 is a tuple of distinct variables.
  add this rule to P, and call the resultant program P1.
  the predicate q_1 is the new query predicate.
  Gen_Prop_predicate-constraints (P1).
  call the resultant program P2.
  Gen_Prop_QRP-constraints (P2).
  delete rules defining q_1 from the resultant program.
  the resultant program P3 is the rewritten program obtained
  by generating
    and propagating minimum QRP-constraints.
}
```

APPENDIX D $P^{qrp,mg}$ VERSUS $P^{mg,qrp}$

We first give an example where using procedure `Gen_Prop_QRP-constraints` on a program P followed by a constraint magic rewriting on the rewritten program is preferable to using constraint magic rewriting on P followed by using procedure `Gen_Prop_QRP-constraints` on the rewritten program.

Example D.1 ($P^{qrp,mg}$ is better than $P^{mg,qrp}$). Consider the following program P:

$$r1: q(X,Y) :- a1(X,Y), X \leq 4.$$
$$r2: a1(X,Y) :- b1(X,Z), a2(Z,Y).$$
$$r3: a2(X,Y) :- b2(X,Y).$$
$$r4: a2(X,Y) :- b2(X,Z), a2(Z,Y).,$$

where q is the query predicate. Each of the rules is range-restricted and hence the bottom-up evaluation of P computes only ground facts. Procedure `Gen_QRP-constraints` would infer the QRP constraint for $a1$ as $\$1 \leq 4$, and the QRP constraint for $a2$ as **true**. Propagating these constraints (using the fold/unfold transformations), we would get the following program P^{qrp}:

$$r1: q(X,Y) :- a1(X,Y), X \leq 4.$$
$$r2: a1(X,Y) :- b1(X,Z), X \leq 4, a2(Z,Y).$$

$$r3{:}a2(X,Y) :- b2(X,Y).$$
$$r4{:}a2(X,Y) :- b2(X,Z), a2(Z,Y).$$

Using constraint magic rewriting on this program (using the query adornment *ff*), we would get $P^{qrp,mg}$:

$$r1{:}\quad q^{ff}(X,Y) :- m\_q^{ff}, a1^{ff}(X,Y), X \leq 4.$$
$$mr1{:}m\_a1^{ff} :- m\_q^{ff}.$$
$$r2{:}\quad a1^{ff}(X,Y) :- m\_a1^{ff}, b1(X,Z), X \leq 4, a2^{bf}(Z,Y).$$
$$mr2{:}m\_a2^{bf}(Z) :- m\_a1^{ff}, b1(X,Z), X \leq 4.$$
$$mr3{:}m\_a2^{bf}(Z) :- m\_a2^{bf}(X), b2(X,Z).$$
$$r3{:}\quad a2^{bf}(X,Y) :- m\_a2^{bf}(X), b2(X,Y).$$
$$r4{:}\quad a2^{bf}(X,Y) :- m\_a2^{bf}(X), b2(X,Z), a2^{bf}(Z,Y).$$

This resultant program is also range-restricted, and hence computes only ground facts.

On the other hand, if constraint magic rewriting were applied on P directly, we would obtain P^{mg}:

$$r1{:}\quad q^{ff}(X,Y) :- m\_q^{ff}, a1^{ff}(X,Y), X \leq 4.$$
$$mr1{:}m\_a1^{ff} :- m\_q^{ff}.$$
$$r2{:}\quad a1^{ff}(X,Y) :- m\_a1^{ff}, b1(X,Z), a2^{bf}(Z,Y).$$
$$mr2{:}m\_a2^{bf}(Z) :- m\_a1^{ff}, b1(X,Z).$$
$$mr3{:}m\_a2^{bf}(Z) :- m\_a2^{bf}(X), b2(X,Z).$$
$$r3{:}\quad a2^{bf}(X,Y) :- m\_a2^{bf}(X), b2(X,Y).$$
$$r4{:}\quad a2^{bf}(X,Y) :- m\_a2^{bf}(X), b2(X,Z), a2^{bf}(Z,Y).$$

On this rewritten program, procedure `Gen_QRP-constraints` would infer the QRP constraint $\$1 \leq 4$ for $a1^{ff}$; the rest of the derived predicates have **true** as their minimum QRP constraint. Propagating this constraint, we would obtain $P^{mg,qrp}$:

$$r1{:}\quad q^{ff}(X,Y) :- m\_q^{ff}, a1^{ff}(X,Y), X \leq 4.$$
$$mr1{:}m\_a1^{ff} :- m\_q^{ff}.$$
$$r2{:}\quad a1^{ff}(X,Y) :- m\_a1^{ff}, b1(X,Z), X \leq 4, a2^{bf}(Z,Y).$$
$$mr2{:}m\_a2^{bf}(Z) :- m\_a1^{ff}, b1(X,Z).$$
$$mr3{:}m\_a2^{bf}(Z) :- m\_a2^{bf}(X), b2(X,Z).$$
$$r3{:}\quad a2^{bf}(X,Y) :- m\_a2^{bf}(X), b2(X,Y).$$
$$r4{:}\quad a2^{bf}(X,Y) :- m\_a2^{bf}(X), b2(X,Z), a2^{bf}(Z,Y).$$

Again this program is range-restricted, and hence computes only ground facts.

Note, however, that the two programs $P^{qrp,mg}$ and $P^{mg,qrp}$ are not equivalent. In particular, rule $mr2$ differs in the two programs. Rule $mr2$ in $P^{qrp,mg}$ is more restrictive than rule $mr2$ in $P^{mg,qrp}$, and hence the evaluation of $P^{qrp,mg}$ computes fewer facts than the evaluation of $P^{mg,qrp}$, in general.

We now give an example where using constraint magic rewriting on P followed by using procedure `Gen_Prop_QRP-constraints` on the rewritten program is preferable to using procedure `Gen_Prop_QRP-constraints` on a program P followed by using constraint magic rewriting on the rewritten program.

Example D.2 ($P^{mg,qrp}$ is better than $P^{qrp,mg}$). Consider the following program P:

$$r1{:}q(X,Y) :- a1(X,Y).$$
$$r2{:}a1(X,Y) :- b1(X,Z), X \leq 4, a2(Z,Y).$$
$$r3{:}a2(X,Y) :- b2(X,Y).$$
$$r3{:}a2(X,Y) :- b2(X,Z), a2(Z,Y).,$$

where q is the query predicate. Procedure `Gen_QRP-constraints` would infer the QRP constraint for $a1$ as **true** and the QRP constraint for $a2$ as **true**. Consequently, P^{qrp} would be the same as P. Using constraint magic rewriting on this program (using the query adornment bf), we would get $P^{qrp,mg}$:

$$r1:\ q^{bf}(X,Y) :- m\_q^{bf}(X), a1^{bf}(X,Y).$$
$$mr1{:}m\_a1^{bf}(X) :- m\_q^{bf}(X).$$
$$r2:\ a1^{bf}(X,Y) :- m\_a1^{bf}(X), b1(X,Z), X \leq 4, a2^{bf}(Z,Y).$$
$$mr2{:}m\_a2^{bf}(Z) :- m\_a1^{bf}(X), b1(X,Z), X \leq 4.$$
$$mr3{:}m\_a2^{bf}(Z) :- m\_a2^{bf}(X), b2(X,Z).$$
$$r3:\ a2^{bf}(X,Y) :- m\_a2^{bf}(X), b2(X,Y).$$
$$r4:\ a2^{bf}(X,Y) :- m\_a2^{bf}(X), b2(X,Z), a2^{bf}(Z,Y).$$

This resultant program is also range-restricted, and hence computes only ground facts.

On the other hand, if constraint magic rewriting were applied on P directly, we would obtain P^{mg}, which is the same as $P^{qrp,mg}$ (because P is the same as P^{qrp}). On this rewritten program, procedure `Gen_QRP-constraints` would infer the QRP constraint $\$1 \leq 4$ for m_a1^{bf}; the rest of the derived predicates have **true** as their minimum QRP constraint. Propagating this constraint, we would obtain $P^{mg,qrp}$:

$$r1:\ q^{bf}(X,Y) :- m\_q^{bf}(X), a1^{bf}(X,Y).$$
$$mr1{:}m\_a1^{bf}(X) :- m\_q^{bf}(X), X \leq 4.$$
$$r2:\ a1^{bf}(X,Y) :- m\_a1^{bf}(X), b1(X,Z), X \leq 4, a2^{bf}(Z,Y).$$
$$mr2{:}m\_a2^{bf}(Z) :- m\_a1^{bf}(X), b1(X,Z), X \leq 4.$$
$$mr3{:}m\_a2^{bf}(Z) :- m\_a2^{bf}(X), b2(X,Z).$$
$$r3:\ a2^{bf}(X,Y) :- m\_a2^{bf}(X), b2(X,Y).$$
$$r4:\ a2^{bf}(X,Y) :- m\_a2^{bf}(X), b2(X,Z), a2^{bf}(Z,Y).$$

Again this program is range-restricted, and hence computes only ground facts.

Again, the two programs $P^{qrp,mg}$ and $P^{mg,qrp}$ are not equivalent. In particular, rule $mr1$ differs in the two programs. Rule $mr1$ in $P^{mg,qrp}$ is more restrictive than rule $mr1$ in $P^{qrp,mg}$, and hence the evaluation of $P^{mg,qrp}$ computes fewer facts than the evaluation of $P^{qrp,mg}$, in general.

REFERENCES

1. Balbin, I., Kemp, D. B., Meenakshi, K., and Ramamohanarao, K., Propagating Constraints in Recursive Deductive Databases, in *Proceedings of the North American Conference on Logic Programming*, October 1989, pp. 16–20.
2. Baudinet, M., Niezette, M., and Wolper, P., On the Representation of Infinite Temporal Data and Queries, in *Proceedings of the Tenth ACM Symposium on Principles of Database Systems*, Denver, Colorado, May 1991, pp. 280–290.
3. Brodsky, A. and Sagiv, Y., Inference of Inequality Constraints in Logic Programs, in *Proceedings of the Tenth ACM Symposium on Principles of Database Systems*, Denver, Colorado, May 1991, pp. 227–240.
4. Chomicki, J., Polynomial Time Query Processing in Temporal Deductive Databases, in *Proceedings of the Ninth ACM Symposium on Principles of Database Systems*, Nashville, Tennessee, April 1990, pp. 379–391.
5. Gardner, P. A. and Shepherdson, J. C., Unfold/Fold Transformations of Logic Programs, in J-L. Lassez and G. Plotkin (Eds.), *Computational Logic*, The MIT Press, Cambridge, MA, 1991.
6. Jaffar, J. and Lassez, J-L., Constraint Logic Programming, in *Proceedings of the 14th ACM POPL*, Munich, January 1987, pp. 111–119.
7. Kanellakis, P. C., Kuper, G. M., and Revesz, P. Z., Constraint Query Languages, in *Proceedings of the Ninth ACM Symposium on Principles of Database Systems*, Nashville, Tennessee, April 1990, pp. 299–313.
8. Lassez, J-L. and Maher, M. J., On Fourier's Algorithm for Linear Arithmetic Constraints, Technical Report, IBM, T. J. Watson Research Center, 1988.
9. Mumick, I. S., Finkelstein, S. J., Pirahesh, H., and Ramakrishnan, R., Magic Conditions, in *Proceedings of the Ninth ACM Symposium on Principles of Database Systems*, Nashville, Tennessee, April 1990, pp. 314–330.
10. Ramakrishnan, R., Magic Templates: A Spellbinding Approach to Logic Programs, in *Proceedings of the International Conference on Logic Programming*, Seattle, Washington, August 1988, pp. 140–159.
11. Revesz, P. Z., A Closed Form for Datalog Queries with Integer Order, in *International Conference on Database Theory*, France, December 1990, pp. 187–201.
12. Sebelik, J. and Stepanek, P., Horn Clause Programs for Recursive Functions, in K. Clark and S.-A. Tarnlund (Eds.), *Logic Programming*, Academic, New York, 1982.
13. Srivastava, D., Subsumption and Indexing in Constraint Query Languages with Linear Arithmetic Constraints, *Ann. Math. Artificial Intelligence* 8(3–4) (1993).
14. Tamaki, H. and Sato, T., Unfold/Fold Transformations of Logic Programs, in *Proceedings of the Second International Conference on Logic Programming*, Uppsala, Sweden, July 1984, pp. 127–138.
15. Tarski, A., *A Decision Method for Elementary Algebra and Geometry*, University of California Press, Berkeley, CA, 1951.
16. Ullman, J. D., *Principles of Database and Knowledge-Base Systems*, Vols. I and II, Computer Science Press, Rockville, MD, 1989.
17. Van Gelder, A., Deriving Constraints among Argument Sizes in Logic Programs, *Ann. Math. Artificial Intelligence* 3:361–392 (1991).