# Approximate Rewriting of Queries Using Views

Foto Afrati[1], Manik Chandrachud[2], Rada Chirkova[2], and Prasenjit Mitra[3]

[1] School of Electrical and Computer Engineering
National Technical University of Athens
Athens, Greece
`afrati@softlab.ntua.gr`

[2] Computer Science Department
NC State University
Raleigh, NC USA
`mnchandr@ncsu.edu`
`chirkova@csc.ncsu.edu`

[3] School of Information Sciences and Technology
Pennsylvania State University
University Park, PA USA
`pmitra@ist.psu.edu`

**Abstract.** We study *approximate,* that is contained and containing, rewritings of queries using views; these rewritings produce respectively a subset and a superset of the set of query answers. We consider conjunctive queries (and views) with arithmetic comparisons (CQACs), which capture the full expressive power of SQL select-project-join queries.
For contained rewritings, we present a sound and complete algorithm *Build-MaxCR* for constructing, for CQAC queries and views, a maximally contained rewriting (MCR) whose all CQAC disjuncts have up to a predetermined number of view literals. (This restriction on the number of view literals is due to the fact that for CQAC queries and views, a *general* view-based CQAC MCR may have an unbounded number of relational subgoals.) For containing rewritings, we present a sound and efficient algorithm *pruned-MiCR,* which computes a CQAC containing rewriting that does not contain any other CQAC containing rewriting (i.e., computes a minimally containing rewriting, MiCR) *and* that has the minimum possible number of relational subgoals. As a result, the MiCR rewriting produced by our algorithm *pruned-MiCR* may be very efficient to execute. Our experimental results indicate that both algorithms have good scalability and perform well in many practical cases, due to their extensive pruning of the search space.

## 1 Introduction

Rewriting queries using views and then executing the rewritings to answer the queries is an important technique used in data warehousing, information integration, query optimization, and other applications, see [1–5] and references therein. Previous research has focused primarily on obtaining equivalent rewritings of queries that can be used to derive the tuples that form *exact* query answers (see, e.g., [6–8]) or maximally contained rewritings (MCRs) that can be used to derive a maximal subset of the set of query answers that can be obtained using the given views (see, e.g., [9, 3, 10–12]). When there does not exist any equivalent or maximally contained rewriting of a user query in terms of the available views, then for such a query, traditional query-processing engines will return no answers at all. However, in many applications, such as querying the World-Wide Web, mass marketing, or searching for clues related to terrorism suspects, users prefer to get a superset of the query answers, rather than getting no answers at all. Another example is peer data-management systems (see, e.g., [13, 14]) used for decentralized data sharing, where no single peer may contain the complete information needed to answer a query. In this scenario, the best strategy may be to obtain an *approximate* query answer using the incomplete information made available by the neighboring peers. In such cases a seriously restricted subset — in addition to a superset — of answers to a query is also acceptable, as these can act as upper and lower bounds [15] on the actual set of answers. (In this paper we restrict our use of the term "approximate" to rewritings that produce subsets (resp. supersets) of the query answers on all databases.) As yet another example, consider a system in which security issues exist. We may want to ensure that a certain class of users do not obtain even a single answer to a secure query. If there is no way for a user to produce a rewriting of a secure query by using the available query language and the given set of views, then that set of views can be considered secure [16] with respect to the user's query language. These examples show there may be several rewritings of interest, other than just equivalent rewritings and MCRs.

In this paper we study *contained* and *containing* rewritings of queries using views, which we refer to collectively as *approximate rewritings.* We focus on conjunctive queries with arithmetic comparisons (CQACs), that is on the language capturing the full expressive power of practically important SQL select-project-join (SPJ) queries. (The well-understood language of conjunctive queries [17] does not capture the in- or non-equalities that are characteristic of SQL SPJ queries.) Specifically, we assume CQAC queries and views, and consider CQAC rewritings, possibly with unions (UCQACs). Note that the well-studied (for conjunctive queries and views) problems of finding equivalent rewritings and MCRs are recognized as being significantly more complex for CQACs, with many practically important cases still unexplored [3, 12].

We now present a motivating example for some of the types of rewritings that we consider in this paper.

*Example 1.* Consider a SQL query `Q` that asks for the agent's name and phone number for all homes with three or more bedrooms that are for sale for less than $500,000.

```
Q:  SELECT AgentName, ContactPhone FROM HomesForSale
    WHERE Price < 500000 AND NumberOfBedrooms >= 3;
```

Suppose that $\mathcal{V} = \{$`V1, V2, V3, V4, V5, V6`$\}$ is the set of views (with straightforward SQL definitions) that can be used to produce rewritings of `Q`. We assume that we can construct rewritings of Q in the (rewriting) language of SQL select-project-join queries *(SQL SPJ),* using views in $\mathcal{V}$.

```
V1: SELECT AgentName, ContactPhone FROM HomesForSale
    WHERE Price < 300000 AND NumberOfBedrooms >= 3;

V2: SELECT AgentName, ContactPhone FROM HomesForSale
    WHERE Price < 250000 AND NumberOfBedrooms >= 3;

V3: SELECT AgentName, ContactPhone FROM HomesForSale
    WHERE Price > 400000 AND Price < 450000 AND NumberOfBedrooms >= 3;

V4: SELECT AgentName, ContactPhone FROM HomesForSale
    WHERE Price < 600000;

V5: SELECT AgentName, ContactPhone FROM HomesForSale
    WHERE NumberOfBedrooms >= 2;

V6: SELECT ContactPhone FROM HomesForSale
    WHERE NumberOfBedrooms >= 2;
```

A rewriting that uses V1 alone[4] is a *contained rewriting* of Q using $\mathcal{V}$, since it may return a proper subset of the query answer, rather than the exact answer. Similarly, each of V2 and V3 is also a contained rewriting. On the other hand, each of V4 and V5 is a *containing rewriting* of Q using $\mathcal{V}$, as it may return a proper superset of the query answer.

Although each of V1, V2 and V3 is a contained rewriting of query Q, only V1 and V3 are *maximally* contained rewritings (MCRs) of Q using $\mathcal{V}$ in our SQL SPJ rewriting language. In this language, V2 is not an MCR because its answers are a subset of the answers to rewriting V1. But in the more expressive rewriting language that would allow us to take unions of SQL SPJ queries, the union of V1 and V3 would be the MCR of Q using $\mathcal{V}$, while V1 or V3 alone would no longer be MCRs. Similarly among containing rewritings, the intersection of V4 and V5 is a more containing rewriting than either V4 or V5 alone. Note that although V6 is very similar to V5, it does not form a containing rewriting, because it does not select the attribute AgentName and hence a rewriting consisting of V6 alone would not be safe. □

We focus on the problem of finding *approximate rewritings* of queries using views, that is, on finding rewritings that are contained in the given query and, separately, rewritings that contain the given query. The approaches in the literature that address the problem of contained rewritings, specifically MCRs [10, 11], have primarily considered conjunctive (CQ) queries without arithmetic comparisons (ACs), and there is very little work on more general cases (see, e.g., [12]). The problems of finding MCRs and minimally-containing rewritings [18–20] in

[4] Each rewriting of Q based on view Vi discussed in this paragraph is Vi itself, e.g., the rewriting based on V1 is query V1.

the presence of arithmetic comparisons remain open in the general case. The complexity of the problems in the presence of ACs is mainly due to the more complex containment test — the containment test is NP-complete in the case of CQs [17] but $\Pi_2^P$-complete [3, 21] in the case of CQACs. We illustrate the challenge by an example from [11].

*Example 2.* Consider CQAC query $Q$ and CQAC view $V$, both defined using binary predicate $p$, as well as a CQAC query $R$ defined in terms of $V$.

$Q()$ :- $p(A,B),\ A \leq B$.
$V()$ :- $p(X,Y),\ p(Y,X)$.
$R()$ :- $V()$.

By the containment tests of [3, 21] it holds that $R$ is a *contained* rewriting of the query $Q$. Observe that the containment cannot be established using a single containment mapping [17] from $Q$ to the expansion of $R$. □

The specific contributions presented in this paper are as follows:

1. **Contained rewritings:** Pottinger and Halevy [11] have developed algorithm MiniCon IP, which efficiently finds UCQAC MCRs for special cases of CQAC queries, views, and rewritings, specifically for those cases where the "homomorphism property" [22, 23] holds between the expansions of the rewritings and the query.[5] At the same time, MiniCon IP cannot find the rewriting $R$ for the problem input of Example 2. We present a sound and complete algorithm called *Build-MaxCR,* for constructing a UCQAC size-limited MCR (that is, an MCR that has up to a predetermined number of view literals) of arbitrary CQAC queries using arbitrary CQAC views. Specifically, when given the query and view of Example 2, Build-MaxCR returns the rewriting $R$ of the example. The size-limit restriction of Build-MaxCR is due to the fact that for CQAC queries and views, a view-based CQAC MCR may have an unbounded number of relational subgoals, see [23] and example 5.
2. **Containing rewritings:** We study the problem of deciding whether there exists a containing rewriting of a given query using a given set of views. Ideally, among the possible containing rewritings, a user would prefer one that is minimally containing, that is, one that contains the fewest false positives. We call such a rewriting the *minimally containing rewriting* (MiCR) of the given query using the given set of views [18–20]. Intuitively, a MiCR is the analog of an MCR — only it produces a superset rather than a subset of the set of query answers, and so it can be thought of as approximating the query from the other direction. A MiCR finds the answers that are in all containing rewritings. Out of all the containing rewritings, a MiCRs gives the highest guarantee of the quality of the answer, by minimizing the number

[5] Intuitively, the *homomorphism property* is said to hold between a query and its rewriting when a single mapping from the query to the expansion of the rewriting is sufficient to establish the containment of the rewriting in the query.

of false positives. Joins in MiCRs increase the cost of their execution; thus minimizing the number of joins in a MiCR for a given query and set of views is important. We give a sound algorithm that finds minimal MiCRs. The algorithm is complete in the special case where the "homomorphism property" [22, 23] holds. The idea of pruned-MiCR is quite general and thus applicable beyond containing rewritings. Specifically, a straightforward modification of pruned-MiCR could be used to reduce the number of relational subgoals of (and thus to provide more efficient execution options for) the outputs of our algorithm Build-MaxCR.

3. **Reducing runtime of containment checking:** Finally, we study the problem of reducing the runtime of containment checking between two CQAC queries, and propose a runtime-reduction technique that takes advantage of some attributes drawing values from disjoint domains. (Intuitively, it does not make sense to compare the values of, e.g., attributes "price" and "name".) This technique can be used in a variety of algorithms. Specifically, it is applicable to our proposed algorithms Build-MaxCR and pruned-MiCR.

Table 1 gives a summary of our results and contributions.

| | **Contained Rewritings** | **Containing Rewritings** |
|---|---|---|
| **Decidability** | UCQAC size-limited MCR for CQACs | UCQACs with negation [18] |
| **Complexity** | CQ: NP [6] | CQAC homomorphism property: NP-complete (Sec. 4.2) |
| **Algorithms** | Size-limited UCQAC MCRs for CQACs | Global minimization of MiCR for CQACs |
| **Previous Work** | MCR [9, 12] | MiCR [18–20] |
| **Applications** | Data warehousing, security, privacy | Mass marketing, P2P, information retrieval |

**Table 1.** Our contributions, previous work, and applications.

Our experimental results indicate that both proposed algorithms perform well in many practical cases, due to their extensive pruning of the search space. In addition, our experiments show good scalability of our algorithms.

In the remainder of this section we review related work. Section 2 reviews the terminology and background results. In Section 3 we present our algorithm Build-MaxCR for finding UCQAC maximally contained rewritings. Section 4 discusses our results on containing rewritings and introduces our algorithm pruned-MiCR for finding efficient to execute formulations of CQAC minimally containing rewritings. In Section 5 we present our proposed optimizations of the containment test for CQAC queries. Section 6 contains our experimental results on the performance and scalability of our algorithms.

## Related Work

The problem of using views in query answering [6] is relevant in applications in information integration [3], data warehousing [9], web-site design [24], and query optimization [5, 6, 25]. Algorithms for finding rewritings of queries using views include the bucket algorithm [19, 26], the inverse-rule algorithm [27–29], the MiniCon algorithm [11], and the shared-variable-bucket algorithm [10]; see [9] for a survey. Almost all of the above work focuses on investigating MCRs or equivalent rewritings [3, 7], as it takes its motivation mostly from information integration and query optimization. Query-rewriting algorithms depend upon efficient algorithms for checking query containment. Existing work on query containment shows that adding arithmetic comparisons to queries and views makes these problems significantly more challenging [30, 22, 21].

Since we consider rewritings that may return false positives or false negatives, our work has similarities with approximate answering of queries using views, see [31–34] and references therein. Approximate query answering is useful when exact answers to the queries cannot be found, and the user would rather have a good-quality approximate answer returned by the system.[6] Lee et al. [35] have considered non-equivalent query rewritings, applied to the problem of maintaining view definitions using a quantitative estimation of the quality of the relaxed query and enabling a tradeoff between performance and the quality of answers. Rather than focusing on performance, our work considers the problem of finding rewritings for the cases where computational and storage resources are not constrained.

The problem of finding containing rewritings of queries using views has been introduced in [19] and also addressed in [18], which deals with answering queries using views via equivalent, contained, and containing rewritings, in the presence of access patterns, integrity constraints, disjunction, and negation. The paper reports complexity results, which render the problem intractable in the general case. The language of rewritings considered in [18] is union of conjunctive queries with negation. Our work focuses on the existence of rewritings in the language of conjunctive queries with arithmetic comparisons (but without negation). Also, in view of the intractability results, we identify special cases where a more efficient algorithm exists for constructing rewritings and outline them.

Other related work includes the results of Rizvi et al. [36], where query-rewriting techniques are used for fine-grained access control, and the work of Miklau et al. [37], which contains a formal probabilistic analysis of information disclosure in data exchange under the assumption of independence among the relations and data in a database. Related work in security and privacy includes [38]. Calvanese et al. [39] have discussed query answering, rewriting, and losslessness with respect to two-way regular path queries. In our work, we concentrate only on query rewritings.

## 2 Preliminaries

In this section, we review some standard concepts related to answering queries using views and introduce some notations that we will use throughout the paper. Some of the definitions are taken from [40, 41].

[6] In our work, we do not measure approximations using probabilities or uncertainty, but the answers to the queries are approximate in the sense that the derived answers may contain false positives.

## 2.1 Queries, Containment, and Views

We consider *conjunctive queries with arithmetic comparisons* (CQACs), that is, select-project-join SQL queries with equality and arithmetic-comparison selection conditions. Each *arithmetic comparison* (AC) subgoal is of the form $X\ \theta\ Y$ or $X\ \theta\ c$,[7] where the comparison operator $\theta$ is one of $<, \leq, >, \geq, \neq$. We assume that database instances are over densely totally ordered domains. A variable is called *distinguished* if it appears in the query head. In the rest of the paper, for a query $Q$ we denote the conjunction of all relational subgoals in $Q$ as $Q_0$ and the conjunction of all ACs in $Q$ as $\beta$. We will use the term *semi-interval CQAC (SI-CQAC)* to refer to conjunctive queries with arithmetic comparisons, where each comparison in the query is either one of $X < c$, $X \leq c$ (*left semi-interval*) or one of $X > c$, $X \geq c$ (*right semi-interval*).

**Definition 1. (Query containment)** *A query $Q_1$ is* contained *in a query $Q_2$, denoted $Q_1 \sqsubseteq Q_2$, if and only if, for all databases $D$, the answer to $Q_1$ on $D$[8] is a subset of the answer to $Q_2$ on $D$, that is, $Q_1(D) \subseteq Q_2(D)$.*

Chandra and Merlin [17] have shown that a CQ $Q_1$ is contained in another CQ $Q_2$ if and only if there exists a *containment mapping* from $Q_2$ to $Q_1$ (*Containment Mapping Theorem*). The mapping maps the head and all the subgoals of $Q_2$ to $Q_1$; it maps each variable to a single variable or constant, and each constant to the same constant. The containment test for CQACs however, is more complicated. There are two ways to test the containment of CQAC $Q_1$ in CQAC $Q_2$ [30, 22]; we will describe them very briefly, for more details see, e.g., [42]. The first test uses the notion of a *canonical database*: For each relational subgoal $p_i(\bar{X}_i)$ in $Q$, a canonical database for $Q$ contains one tuple $t$ in the base relation $p_i$, such that $t$ is the list of "frozen" variables and constants from $\bar{X}_i$ (i.e., in forming $t$ each variable in $\bar{X}_i$ is "frozen" to a unique constant except that equated variables are frozen to the same constant and each constant in $\bar{X}_i$ is kept as it is). We define one canonical database for each total ordering of the variables and constants in $Q_1$ that satisfies the ACs in $Q_1$. The test says that a $Q_1$ is contained in $Q_2$ if and only if $Q_2$ computes, on all the canonical databases of $Q_1$, the same head tuple(s) as the head tuple(s) of $Q_1$.

The second containment test is as follows:

**Theorem 1.** *$Q_1 \sqsubseteq Q_2$ if and only if the following logical implication $\phi$ is true:*

$$\phi : \beta_1' \Rightarrow \mu_1(\beta_2') \vee \ldots \vee \mu_k(\beta_2')$$

*where $\mu_i$'s are all containment mappings from $Q_2'$ to $Q_1'$ and $\beta_i'$ is a conjunction of all ACs in $Q_i'$. That is, the ACs in the normalized query[9] $Q_1'$ logically imply*

[7] We use uppercase letters to denote variables and lowercase letters for constants.

[8] $Q_1(D)$ denotes the set of answer tuples obtained by evaluating query $Q_1$ on database $D$

[9] An equivalent '*normalized version* [22, 40] of a CQAC query $Q$ does not have constants or repetitions of variable names in relational subgoals and has compensating built-in equality conditions.

*(denoted "$\Rightarrow$") the disjunction of the images of the ACs of the normalized query $Q'_2$ under each mapping $\mu_i$.*

If there exists a containment mapping $\mu_i$ such that the right-hand side of $\phi$ is reduced to only one $\mu_i(\beta'_2)$, we say the *homomorphism property* holds. Afrati, et al., [42] have shown that when the homomorphism property holds, the implication can be checked directly on queries without normalizing them. Checking CQAC containment is less complex in that case, because we need to check for the existence of just one mapping that satisfies the implication.

## 2.2 Rewriting Queries using Views

We consider the problem of finding rewritings under the closed-world assumption [7] (where the views are both sound and complete, i.e., for a given database, each view instance stores exactly the tuples satisfying the view definition), as well as under the open-world assumption [7, 26] (where the views are sound but not necessarily complete, i.e., a view instance might store only some of the tuples satisfying the view definition).

Suppose that we are looking for an answer to query $Q$ on database $D$ and that our access to $D$ is defined through a set of views $\mathcal{V} = \{V_1, \ldots, V_m\}$. So instead of directly evaluating $Q$ on $D$, we first rewrite $Q$ in terms of $\mathcal{V}$, and then evaluate this rewriting on $D_{\mathcal{V}}$, that is, the database with schema $\mathcal{V}$.

We consider the following types of rewritings $R$ of query $Q$ using views $\mathcal{V}$:

**Definition 2. (Rewritings)**

1. *a. (CWA) $R$ is a* contained rewriting *of $Q$ using $\mathcal{V}$ under the CWA if and only if $R(D_{\mathcal{V}}) \subseteq Q(D)$ for all databases $D$.*
   *b. (OWA) $R$ is a* contained rewriting *of $Q$ using $\mathcal{V}$ under the OWA if and only if $R(I_{\mathcal{V}}) \subseteq Q(D)$ for all databases $D$ and view instances $I_{\mathcal{V}}$ such that $I_{\mathcal{V}} \subseteq D_{\mathcal{V}}$.*
2. *(CWA) $R$ is a* containing rewriting *of $Q$ using $\mathcal{V}$ if and only if $Q(D) \subseteq R(D_{\mathcal{V}})$ for all $D$.*
3. *(CWA) $R$ is an* equivalent rewriting *of $Q$ using $\mathcal{V}$ if and only if $Q(D) = R(D_{\mathcal{V}})$ for all $D$.*

Since the result of a containing rewriting must contain all tuples that occur in the answer to $Q$, containing rewritings make sense only when the views that are used in constructing the containing rewriting are complete. Hence, containing rewritings are considered only under the CWA and not under the OWA. The same is true for equivalent rewritings, since an equivalent rewriting of $Q$ is a rewriting that is a contained as well as a containing rewriting of $Q$. However, since the result of a contained rewriting is allowed to leave out some of the answers to $Q$, contained rewritings make sense under the CWA and under the OWA.

Given a query $Q$ and a set of views $\mathcal{V}$, for deciding whether there exists a contained (or containing) rewriting of $Q$ using $\mathcal{V}$, we need to know the language in which we are allowed to construct rewritings. In the rest of the paper, we will assume, unless otherwise stated, that the language of the rewritings for the existence problem is UCQACs.

We define the expansion of a rewriting as follows:

**Definition 3. (Expansion of a rewriting)** *For a CQAC rewriting $R$ that is expressed in terms of CQAC views $\mathcal{V}$, an* expansion *$R^{exp}$ of $R$ is obtained by replacing each view subgoal in $R$ by the all the subgoals in the definition of that view. Each existentially quantified variable in the definition of a view in $R$ is replaced by a unique variable in $R^{exp}$. For a UCQAC rewriting, the expansion is the union of the expansions of the CQACs that occur in that UCQAC.*

The evaluation of contained rewritings cannot return false positives, the evaluation of containing rewritings cannot return false negatives, and the evaluation of equivalent rewritings cannot return either false positives or false negatives. We will use the term *rewriting* to mean a contained or a containing rewriting; we will specify the type whenever it is not obvious from the context.

Theorem 2 is based on definitions 2 and 3 and gives the tests for determining whether a CQAC rewriting $R$ is contained (or containing) rewriting of a CQAC query $Q$ using CQAC views $\mathcal{V}$..

**Theorem 2.** *Let $Q, V_1, \ldots, V_m$ be CQAC views that are expressed in terms of the base relations, and let $R$ be a CQAC rewriting of a CQAC query $Q$, such that $R$ is expressed in terms of $V_1, \ldots, V_m$. Then*

1. *$R$ is a contained rewriting of $Q$ iff $R^{exp} \sqsubseteq Q$.*
2. *$R$ is a containing rewriting of $Q$ iff $Q \sqsubseteq R^{exp}$.*

Contained rewritings are defined under the CWA as well as the OWA. But ascertaining whether a given rewriting is a contained rewriting or not, involves only taking the expansion of the rewriting and then checking whether or not this expansion is contained in the query. Since the expansion is the same under the CWA and the OWA, we have the following proposition:

**Proposition 1.** *For queries and views that are CQACs, a UCQAC rewriting is a contained rewriting under the open-world assumption (OWA) iff it is a contained rewriting under the closed-world assumption (CWA).*

## 2.3 Canonical Databases

Next, we introduce some definitions that are necessary for the description of the Build-MaxCR algorithm that we present in Section 3. We begin by recalling the notion of the *expansion of a safe CQAC query.* (All the queries that we consider in the paper are *safe,* i.e., each of their head variables and each variable used in the query ACs also appears in at least one relational subgoal of the query.) Let $\mathcal{V}$ be a set of CQAC views defined on a database schema $\mathcal{P}$, and let $Q$ be a CQAC query defined on schema $\mathcal{V}$. The *expansion of $Q$* is a CQAC query $Q^{exp}$ that is obtained from $Q$ as follows. For each subgoal $v(x_1, \ldots, x_r)$ of $Q$, such that

$v(x_1, \ldots, x_r)$ corresponds to view $V(y_1, \ldots, y_r) \in \mathcal{V}$, replace $v(x_1, \ldots, x_r)$ in the body of $Q$ by the definition of $V$. In each replacement, (i) if $r > 0$ then each $y_i$, $1 \leq i \leq r$, is replaced by $x_i$, and (ii) all nonhead variables in the definition of $V$ are renamed consistently into fresh variables. By construction, query $Q^{exp}$ is defined on the database schema $\mathcal{P}$.

**Definition 4. (Consistent) assignment mapping)** *Let $Q$ be a CQ query defined on a database schema $\mathcal{P}$, and let $D$ be a database with schema $\mathcal{P}$. A mapping $\lambda$ is an* assignment mapping *from $Q$ to $D$ if $\lambda$ maps each variable or constant in the body of $Q$ to a stored value in $D$. (We refer to stored values in databases as* constants.*) An assignment mapping $\lambda$ from $Q$ to $D$ is* consistent *if (i) for each variable $X$ of $Q$, $\lambda$ maps all occurrences of $X$ into the same constant in $D$, and $\lambda$ maps each constant in $Q$ into the same constant in $D$, and (ii) $\lambda$ induces a mapping from the relational subgoals of $Q$ to the tuples stored in $D$ in such a way that for each relational subgoal $p(\ldots)$ of $Q$ with relation name $P$, the image of $p(\ldots)$ under $\lambda$ is a tuple in relation $P$ in $D$.*

For the next definition (Definition 5), we will need the notion of a *canonical database $D_Q$ for a safe CQ query $Q$. $D_Q$* is defined as the result of "freezing" the body of $Q$, which turns each subgoal of $Q$ into a fact in the database. That is, the "freezing" procedure replaces each constant in the body of $Q$ by *the same* constant, and each variable in the body of $Q$ by a distinct constant that is different from all constants in the body of $Q$. The resulting subgoals are the only tuples in the canonical database $D_Q$.

**Definition 5. (Canonical databases for a safe CQAC query; total order on a canonical database for a CQAC query)** *The* set $\mathcal{D}^Q$ of canonical databases for a conjunctive query with arithmetic comparisons (CQAC) *$Q$ is constructed by taking the following steps.*

1. *Treat all the relational subgoals and all the equality ACs of $Q$ as a* conjunctive *query $\bar{Q}$, and then build the canonical database $D_{\bar{Q}}$ of $\bar{Q}$, with the modification that each variable in the body of $\bar{Q}$ is replaced by* the same variable, *to be replaced by numerical constants at the stage of outputting individual canonical databases of the CQAC $Q$. (That is, we use $D_{\bar{Q}}$ as a template for constructing the canonical databases of $Q$, by instantiating the variables in $D_{\bar{Q}}$ by numerical constants.)*
2. *To produce each canonical database of $Q$, consider the set $W$ of constants and variables in the body of $Q$ (with the exception of all non-numerical constants coming from the body of $Q$) as belonging to a totally ordered set, e.g., the integers or reals. The set $\mathcal{D}^Q$ of canonical databases of $Q$ is constructed (starting from the empty set, $\mathcal{D}^Q = \emptyset$) in two stages.*
   *The* **first stage** *generates the set $\mathcal{S}$ of all total orders on $W$, where each total order $S \in \mathcal{S}$ is constructed as follows:*
   (a) *partition all of $W$ into "sets of equal value" (e.g., can the values of variables $X \in W$ and $Y \in W$ be equal in a canonical database), making sure that no set of equal value contain two or more distinct numerical constants from $W$; then*

*(b) to each set $s$ of equal value that contains a single numerical constant $c \in W$, pre-assign to $s$ the value $c$; then*

*(c) among the sets of equal value in the partition, determine the relative order $\mathcal{O}$ of their values, by using the less-than operator and by taking the above pre-assignments into account (i.e., if set $s_1$ has been pre-assigned value $w_1$ and set $s_2$ has been pre-assigned value $w_2 > w_1$, then the relative order in $\mathcal{O}$ of the values of $s_1$ and $s_2$ must imply the inequality "value of $s_1$ is less than the value of $s_2$"); finally,*

*(d) based on the relative order $\mathcal{O}$ of the sets of equal value in the partition, to each set of equal value in the partition assign a numerical (real-number) value that agrees with the relative order $\mathcal{O}$ and such that for each set that has been pre-assigned a value, the final assignment retains the pre-assigned value.*

*The above assignment of numerical values to the sets in the partition of $W$ induces a (consistent) mapping $M$ from the variables of $Q$ (and thus from the variables in the canonical-database template $D_{\bar{Q}}$) to the set of these numerical values. We call each such mapping $M$ a* total-order mapping on a canonical database for query $Q$.

*The* total order *$S$ associated with $M$ is a set of ACs involving all and only the variables and constants of $W$, such that $S$ expresses exactly the conjunction of all the facts (I) and (II) about $\mathcal{O}$, as follows: (I) For each (unordered) pair $(w_1, w_2)$ of variables and/or constants within each set of equal values in $\mathcal{O}$, we have that $w_1 = w_2$, and (II) For each pair $(w_1, w_2)$ of variables and/or constants where $w_1$ and $w_2$ belong to different sets of equal value in $\mathcal{O}$, we have that $w_1 < w_2$ if and only if the respective sets of equal value are in the less-than relationship in $\mathcal{O}$.*

*The* **second stage** *of constructing the canonical databases of $Q$ adds to $\mathcal{D}^Q$ one database $D_S$ for the association of each total order $S \in \mathcal{S}$ with the canonical-database template $D_{\bar{Q}}$. Specifically, canonical database $D_S$ results from applying the mapping $M$ associated with $S$ to the variables in $D_{\bar{Q}}$. To retain the association between the variables of the query $Q$ and the constants in $D_S$, we assume that each canonical database $D_S$ of $Q$ is associated explicitly both with the total-order mapping $M$ and with its associated total order $S$. (Note that applying the total-order mapping $M$ for $D_S$ to the total order $S$ for $D_S$ results in a conjunction of true ACs on the constants of $D_S$.)*

**Definition 6. (Homomorphism property)** *[23] Let $\mathcal{Q}_2$, $\mathcal{Q}_1$ be two classes of CQAC queries. We say that containment testing on the pair ($\mathcal{Q}_2$, $\mathcal{Q}_1$) has the homomorphism property if for any pair of queries ($Q_2$,$Q_1$) with $Q_2 \sqsubseteq \mathcal{Q}_2$ and $Q_1 \sqsubseteq \mathcal{Q}_1$, the following holds: $Q_1 \sqsubseteq Q_2$ iff there is a homomorphism $\mu$ from core($Q_2$) to core($Q_1$) such that $AC(Q1) \Rightarrow (AC(Q2))$.*

Although the homomorphism property is defined for two classes of queries, it is often referred to in the context of holding for two queries, that is, the case in which each of the two classes contain exactly one query. For testing the containment of CQAC $Q_1$ in CQAC $Q_2$, if the homomorphism property holds, then the mapping (homomorphism) from $Q_2$ to $Q_1$ is the containment mapping, say $\mu$, and the right-hand side of the implication in Theorem 1 is reduced to $\mu(\beta_2')$. In such a case, the normalization step in the CQAC containment test is not necessary, and for this special case, the problem of checking CQAC containment is in NP.

**Definition 7. (Single-mapping contained rewriting)** *A CQAC contained rewriting $R$ of a CQAC query $Q$ using a set of CQAC views $\mathcal{V}$ is a* single-mapping contained rewriting *iff the homomorphism property holds for the containment of $R^{exp}$ in $Q$.*

**Definition 8. (Multiple-mapping contained rewriting)** *A CQAC contained rewriting $R$ of a CQAC query $Q$ using a set of CQAC views $\mathcal{V}$ is a* multiple-mapping contained rewriting *iff the homomorphism property does not hold for the containment of $R^{exp}$ in $Q$.*

The rewriting $R$ in Example 3 below is a single-mapping MCR of $Q$ using $V$, while the $R$ in Example 4 is a multiple-mapping MCR.

*Example 3.*
$Q()$ :- $p(A),\ A > 3.$
$V()$ :- $p(X),\ X > 3.$
$R()$ :- $V().$

*Example 4.*
$Q()$ :- $p(A,B),\ A \leq B.$
$V()$ :- $p(X,Y),\ p(Y,X).$
$R()$ :- $V().$

## 3 The Build-MaxCR Algorithm: Finding MCRs for CQACs

In this section we present a sound and complete algorithm called *Build-MaxCR,* for constructing a UCQAC size-limited MCR (that is, an MCR that has up to a predetermined number of view literals) of arbitrary CQAC queries using arbitrary CQAC views. We present and discuss the pseudocode and then formulate the correctness (i.e., soundness and completeness) results for the algorithm.

### 3.1 The Setting and Definitions

**Why Look at *k*-Bounded Rewritings**

In the general case, for CQAC queries and views, the MCR is a union of CQACs. So the next natural question, is whether the number of CQACs in such a UCQAC answer is always bounded? Example 5 below (based on the ideas from [23]) shows, that the UCQAC-MCR may sometimes have an unbounded number of CQACs in it.

*Example 5.*
$Q()$ :- $p(X,Y),\ p(Y,Z),\ s(Y),\ X \geq 2,\ Z \leq 7$
$V_1(L,M)$ :- $p(L,M),\ L \geq 2,\ M \leq 7$
$V_2(A,C)$ :- $p(A,B),\ p(B,C),\ s(A),\ s(C)$

$R_3()$ :- $V_1(L_1,A_1),\ V_2(A_1,C_1),\ V_1(C_1,M_2)$

$R_3^{exp}()$ :- $p(L_1,A_1),\ p(A_1,B_1),\ s(A_1),\ p(B_1,C_1),\ p(C_1,M_2),$
$s(C_1),\ L_1 \geq 2,\ A_1 \leq 7,\ C_1 \geq 2,\ M_2 \leq 7$

$R_4()$ :- $V_1(X,T_1),\ V_2(T_1,T_2),\ V_2(T_2,T_3),\ V_1(T_3,Z)$

$R_4^{exp}()$ :- $p(X,T_1),\ p(T_1,U_1),\ s(T_1),\ p(U_1,T_2),\ p(T_2,U_2),\ s(T_2),$
$p(U_2,T_3),\ p(T_3,Z),\ s(T_3),\ X \geq 2,\ T_1 \leq 7,\ T_3 \geq 2,\ Z \leq 7$

For Example 5, the rewriting $R_3$ is a multiple-mapping contained rewriting of $Q$ using views $V_1$ and $V_2$. However, it is also possible to have another rewriting $R_4$ which is a contained rewriting of $Q$. In fact, starting with $R_3$, which has 3 relational subgoals – the two $V_1$'s at the two ends with one $V_2$ between them – we can obtain $R_4$ by introducing an additional $V_2$ in the middle. Similarly, we can also construct $R_5$, $R_6$, and so on, by repeatedly pumping copies of $V_2$ in the chain of $V_2$'s enclosed between the two $V_1$'s at the two ends. In general, for any $n \geq 3$, $R_n$ is not contained in $R_i$ for any $3 \leq i < n$. Therefore, a UCQAC contained rewriting of $Q$ must include every $R_n$.

Thus Example 5 shows that the number of CQACs in the UCQAC-MCR of a CQAC $Q$ using CQAC $\mathcal{V}$ may not be bounded. Hence an algorithm for finding the UCQAC-MCR may not terminate on some inputs. To ensure that our algorithm Build-MaxCR always terminates, we needed to introduce the concept of size-limited MCRs i.e., $k$-limited MCRs for a user-specified $k$. If UCQAC $R$ is a $k$-limited MCR of $Q$ using $\mathcal{V}$, then any CQAC contained rewriting of $Q$ using $\mathcal{V}$ that has up to $k$ subgoals in it, is guaranteed to be contained in some rewriting in the union $R$. In Example 5, a 5-limited MCR of $Q$ using $\mathcal{V}$ would contain $R_3 \bigcup R_4 \bigcup R_5$. For any user-specified $k$, the Build-MaxCR algorithm is guaranteed to find the $k$-limited MCR.

We begin by defining the problem of constructing a UCQAC size-limited MCR for a CQAC query using CQAC views. We use the following definition:

**Definition 9. (A $k$-bounded (CQAC, UCQAC) query)** *Given a database schema $\mathcal{V}$ and a positive integer number $k$. (1) A CQAC query $Q$ defined on $\mathcal{V}$ is a* $k$-bounded (CQAC) query using $\mathcal{V}$ *if, for the number $n$ of relational subgoals of $Q$, we have $n \leq k$. (2) $Q = \bigcup_i Q_i$ is a* $k$-bounded UCQAC query using $\mathcal{V}$ *if each CQAC component $Q_i$ of $Q$ is a $k$-bounded query using $\mathcal{V}$.*

Now, for the problem of constructing a UCQAC size-limited (specifically $k$-bounded) MCR for a CQAC query using CQAC views,

– the *problem input* is a triple $(Q, \mathcal{V}, k)$, where $Q$ is a CQAC query, $\mathcal{V}$ is a finite set of CQAC views, and $k$ is a natural number;

– the *problem output* is a UCQAC query $P = \bigcup_j P'_j$ defined in terms of $\mathcal{V}$, such that $P^{exp}$ is contained in $Q$, and such that $P$ is a $k$-bounded (UCQAC) query in terms of $\mathcal{V}$.

Our proposed algorithm Build-MaxCR solves the above problem for arbitrary inputs $(Q, \mathcal{V}, k)$ as defined in the preceding paragraph. Our soundness and completeness results for Build-MaxCR, Theorems 3 and 4, establish that for each such input $(Q, \mathcal{V}, k)$, Build-MaxCR returns a maximally contained rewriting of $Q$ in the language of $k$-bounded UCQAC queries over $\mathcal{V}$, if such a rewriting exists. Examples 2 and 6 provide specific illustrations of this general description of how the algorithm works.

## 3.2 Algorithm Build-MaxCR: The Pseudocode and Description

We now discuss the pseudocode for Build-MaxCR, please see Algorithm 1. The general idea of the algorithm is to do a complete enumeration of the CQ parts, call them $\bar{P}_j$, of $k$-bounded CQAC queries defined on schema $\mathcal{V}$. (For a CQAC query $R$, we use the term "CQ part of $R$" to refer to the join of all relational subgoals of $R$, taken together with all the equality ACs implied by $R$.) For each such $\bar{P}_j$, the algorithm associates with $\bar{P}_j$ a *minimum* set $S'_j$ of inequality/nonequality ACs on the variables and constants of $\bar{P}_j$,[10] such that $S'_j$ ensures containment of $\bar{P}_j^{exp} \& S'_j$ in $Q$. The output for Build-MaxCR is the union $P$ of all the CQAC queries $\bar{P}_j \& S'_j$ for which the containment holds. (By [22], $\bar{P}_j^{exp} \& S'_j \sqsubseteq Q$ for each $j$ ensures $P^{exp} \sqsubseteq Q$, where $P = \bigcup_j \bar{P}_j \& S'_j$.)

We now provide a detailed discussion of the pseudocode. Algorithm Build-MaxCR uses an efficient way of enumerating the CQ queries $\bar{P}_j$: The algorithm first enumerates all *cross products*, call them $P_i$, of up to $k$ relational subgoals in terms of $\mathcal{V}$. We call each $P_i$, with $t \leq k$ subgoals, a "CQAC-rewriting template (for $Q$) of size $t$". Note that each $P_i = v_{i1}(\bar{x}_1) \times \ldots \times v_{it}(\bar{x}_t), t \leq k$, is associated with exactly one multiset $\{\{v_{i1}, \ldots, v_{it}\}\}$ of names of views in $\mathcal{V}$, where we consider the set $\mathcal{MS}$ of all such multisets whose sizes do not exceed $k$, and vice versa (i.e., 1:1 association) from $\mathcal{MS}$ to $\{P_i = v_{i1}(\bar{x}_1) \times \ldots \times v_{it}(\bar{x}_t), t \leq k\}$.

For the remaining discussion, we will need the notion of a MaxCR canonical database, and a generative assignment mapping from a CQAC query to MaxCR canonical database.

**Definition 10. (MaxCR canonical database for CQAC query $Q$ and CQAC-rewriting template $P$)** *The set $\mathcal{D}_P^Q$ of* MaxCR canonical databases *for $Q$ and $P$ is constructed in the same way as the set $\mathcal{D}^{(P^{exp})}$ of canonical databases of the expansion $P^{exp}$ of $P$. The only difference is that the set $W$ of constants and variables in the body of $P^{exp}$ ($W$ is used in the construction of*

[10] In this paper we use the term "minimum set of ACs", for a given objective (such as ensuring containment of a query in another query), to refer to any set $S$ of ACs such that the closure of $S$ under composition is a *minimum-size* set that meets the objective.

**Algorithm 1:** Algorithm Build-MaxCR

```
Input   : CQAC query $Q$; set of CQAC views $\mathcal{V}$; $k \in \mathbf{N}$
Output : $k$-bounded UCQAC query $P = \bigcup_j P'_j$ in terms of $\mathcal{V}$ s. t. $P^{exp} \sqsubseteq Q$
1. $P \leftarrow \emptyset$; //$P$ is a union of CQACs in the output of Build-MaxCR
2. for $t = 1$ to $k$ do
    3. $\mathcal{P}_t \leftarrow$ set of all CQAC-rewriting templates of size $t$;
    4. while $\mathcal{P}_t \neq \emptyset$ do
        5. $P_i \leftarrow$ one template from $\mathcal{P}_t$;  $\mathcal{P}_t \leftarrow \mathcal{P}_t - \{P_i\}$;
        6. $\mathcal{D}_i \leftarrow$ the set of all MaxCR canonical databases for $P_i$ and $Q$ that
        make the head of $P_i^{exp}$ true;
        7. $\mathcal{M} \leftarrow$ set of all mappings $\mu_{ij}$ from relational
         subgoals of $Q$ to same-name subgoals of $P_i^{exp}$;
        8. if $\mathcal{M} = \emptyset$ then continue;
        //Single-mapping processing:
        9. for $j = 1$ to $|\mathcal{M}|$ do
            10. produce $\mathcal{D}_{ij} \subseteq \mathcal{D}_i$, where each $D \in \mathcal{D}_i$
                is included in $\mathcal{D}_{ij}$ iff embedding $Q$ in
                $D$ using $\mu_{ij}$ makes the head of $Q$ true;
            11. $S_{ij} \leftarrow$ set of summary ACs for $\mathcal{D}_{ij}$;  //$S_{ij}$ is
                //the logical OR of all total orders for $\mathcal{D}_{ij}$;
            12. $\bar{x}_{ij} \leftarrow \tilde{\mu}_{ij}(headVars(Q))$; // $\mu_{ij}$ induces $\tilde{\mu}_{ij}$
            13. if all variables of $S_{ij}$ also occur in $P_i$ then
                14.$P \leftarrow P \bigcup (P'_{ij}(\bar{x}_{ij})$ :- $P_i \& S_{ij})$;
        //Multiple-mapping processing:
        15. if for some set $J = \{j^{(1)}, \ldots, j^{(r)}, \ldots, j^{(m)}\}$ of the $j$'s above, s.t.
        $m > 1$, all the $\bar{x}_{ij^{(r)}}$'s are the same, call them $\tilde{x}_i$, then
            16. $\tilde{D}_i \leftarrow \bigcup_{r=1}^{m} \mathcal{D}_{ij^{(r)}}$; $\tilde{S}_i \leftarrow$ summary ACs for $\tilde{D}_i$;
            17. if all variables used in $\tilde{S}_i$ also occur in $P_i$ then
                18.$P \leftarrow P \bigcup (P'_i(\tilde{x}_i)$ :- $P_i \& \tilde{S}_i)$;
19. Return $P$.
```

*$\mathcal{D}^{(P^{exp})}$) is extended, for the construction of $\mathcal{D}_P^Q$, to include also all the numerical constants of the query $Q$.*

**Definition 11. (Generative assignment mapping from CQAC query to MaxCR canonical database)** *Given a CQAC-rewriting template $P$ for CQAC query $Q$; let $P^{exp}$ be the expansion of $P$. A* generative assignment mapping from $P^{exp}$ to a MaxCR canonical database $D$ for $Q$ and $P$ *is the total-order mapping $M$ (from the body variables of $P^{exp}$ to numerical constants in $D$) associated with $D$. (See Definition 5 for the details on the mapping $M$.)*

Intuitively, we call $M$ a *generative* assignment mapping because it induces a "generative" mapping $\mu$ from the relational subgoals of $P^{exp}$ to the tuples in $D$. That is, $\mu$ associates each relational subgoal $p$ of $P^{exp}$ with the tuple $t$ of $D$ such that $t$ was generated from $p$ when database $D$ was generated from $P^{exp}$.

Consider a fixed CQAC-rewriting template $P_i$, with $t \leq k$ subgoals. (Build-MaxCR considers all the $P_i$'s exhaustively in the increasing order of $t$.) Build-MaxCR uses $P_i$ to generate *all* $t$-bounded CQAC queries $P'_j = P_j \& S'_j$ in terms of $\mathcal{V}$, such that the CQ part ($\bar{P}_j$) of each $P'_j$ corresponds to *the same* multiset of names of views as $P_i$'s multiset. (The difference between the CQ parts of the different $P'_j$'s for a fixed $P_i$ is in the equality ACs that are associated with each $P'_j$ and that equate some or all of the variables in the cross product $P_i$.) Moreover, Build-MaxCR generates all the $P'_j$'s for $P_i$ by considering *only once* and *only one* set of essentially canonical databases for $P_i^{exp}$. This set of databases, which we call "MaxCR canonical databases for $P_i$ and $Q$" (Definition 10), is used by Build-MaxCR to compute the (both equality and inequality/nonequality) ACs in the definition of each rewriting $P'_j$ that Build-MaxCR outputs based on $P_i$. More precisely, Build-MaxCR uses the MaxCR canonical databases both for computing the ACs for each $P'_j$ (each set of computed ACs is the total order for one of the MaxCR databases, see Definitions 5 and 10) and for ensuring the positive outcome for the containment test for $(P'_j)^{exp} \sqsubseteq Q$. The containment test, which is implicit in the algorithm (i.e., the outcome of the test is positive by construction of $P'_j$, under certain conditions, which are all that Build-MaxCR needs to check to ensure the outcome of the containment test; checking the conditions takes linear time in the size of $P'_j$), is based on the same (MaxCR) canonical databases as for the above AC computation. We now consider the entire construction in detail.

We begin by specifying the set, call it $\mathcal{C}_i$, of *MaxCR canonical databases for $P_i$ and $Q$,* where $P_i$ is a CQAC-rewriting template for $Q$ of size $t \leq k$. (For a formal specification, see Definition 10.) The basis for the definition of MaxCR canonical databases is the (standard) notion of the set $\mathcal{C}'_i$ of canonical databases for $P_i^{exp}$, see Definition 5. By definition, each database $C' \in \mathcal{C}'_i$ is constructed using one total order, $S'$, on the set $VarsConsts(P_i^{exp})$ of variables and constants in the body of $P_i^{exp}$. When generating the set $\mathcal{C}_i$ of *MaxCR* canonical databases for $P_i$ and $Q$, we use exactly the same (i.e., the standard) database-construction process as for $\mathcal{C}'_i$, except that we generate all the total orders based on the set $MaxCrVars = VarsConsts(P_i^{exp}) \bigcup Consts(Q)$, where $Consts(Q)$ is the set of all constants in the definition of the input query $Q$. A *total order* associated with each $C \in \mathcal{C}_i$ is the set of ACs on $MaxCrVars$ that yields database $C$ in the construction of $\mathcal{C}_i$. As illustrated in Example 6 and as shown in the proof of Theorem 4, using the constants of $Q$ in the construction of $\mathcal{C}_i$ ensures completeness of the algorithm Build-MaxCR.

Algorithm Build-MaxCR works with a specific subset $\mathcal{D}_i$ of all MaxCR canonical databases $\mathcal{C}_i$ for $P_i$ and $Q$. Let us treat the CQAC-rewriting template $P_i$ as a CQ query without nondistinguished variables. Then $\mathcal{D}_i$ is defined as the largest subset of $\mathcal{C}_i$ such that each $D \in \mathcal{D}_i$ *makes the head of the query* $P_i^{exp}$ *true* – that is, $P_i^{exp}$ returns a nonempty answer on $D$. As shown in Example 6, the presence of ACs in the body of $P_i^{exp}$ may prevent $P_i^{exp}$ from returning a nonempty answer on some databases in $\mathcal{C}_i$, thus rendering $\mathcal{D}_i$ a proper subset of $\mathcal{C}_i$. (The presence of ACs in the body of $P_i^{exp}$ is due to the expansion of the view literals when building $P_i^{exp}$ from $P_i$, recall that the views in $\mathcal{V}$ in the Build-MaxCR input are defined in the language of CQAC queries.)

Once Build-MaxCR has computed the set $\mathcal{D}_i$ of MaxCR canonical databases for $P_i$ and $Q$ such that each $D \in \mathcal{D}_i$ makes the head of $P_i^{exp}$ true, the algorithm next determines which of the databases in $\mathcal{D}_i$ also make true the head of the input query $Q$, under specific restrictions to be detailed in the next paragraph. The purpose of this stage is as follows. Suppose some subset $\mathcal{D}_{ij}$ of $\mathcal{D}_i$ makes true, in the above sense, the heads of both $P_i^{exp}$ and $Q$. Algorithm Build-MaxCR comes up with a set $S_{ij}$ of *summary ACs* for this set $\mathcal{D}_{ij}$ of MaxCR canonical databases for $P_i$ and $Q$. That is, $S_{ij}$ is a set of ACs that characterizes exactly the databases in $\mathcal{D}_{ij}$, in the sense that $S_{ij}$ is the logical OR of the total-order ACs for the set $\mathcal{D}_{ij}$. (Please see Example 6 for an illustration.) Then Build-MaxCR considers the CQAC conjunction $P'_{ij} = P_i \& S_{ij}$. Suppose $P'_{ij}$ satisfies the safety condition, that is, all the variables that are used in $S_{ij}$ are head variables of $P_i$. In this case, by Theorem 3 we have that $(P'_{ij})^{exp}$ is contained in $Q$. (The head variables of $P'_{ij}$ are determined by Build-MaxCR in the process of constructing $\mathcal{D}_{ij}$.) Thus, Build-MaxCR adds $P'_{ij}$ to the union $P$ of $k$-bounded CQAC queries that $(P)$ will be eventually output by the algorithm.

We now provide the details, by explaining the restrictions under which Build-MaxCR produces the sets $\mathcal{D}_{ij}$ of MaxCR databases for $P_i$ and $Q$, such that each set $\mathcal{D}_{ij}$ makes the heads of both $P_i^{exp}$ and $Q$ true. Recall that $\mathcal{D}_i$ is the set of MaxCR canonical databases for $P_i$ and $Q$ such that each database in $\mathcal{D}_i$ makes the head of $P_i^{exp}$ true. For each database $D \in \mathcal{D}_i$, let us denote by $\iota$ the mapping that associates each relational subgoal $r$ of $P_i^{exp}$ with the stored tuple $t \in D$ such that $t$ is generated from $r$ by construction of database $D$. Let us call $\iota$ the "generative" mapping from $P_i^{exp}$ to $D$. (See Definition 11 and its discussion.) Note that $\iota$ induces a (consistent, in fact generative, see Definition 11) assignment mapping $\tilde{\iota}$ from the variables of $P_i^{exp}$ to the stored values in $D$, with the property that $\tilde{\iota}(headVars(P_i^{exp}))$ is always an answer to $P_i^{exp}$ on the database $D$, by construction of the MaxCR canonical database $D$ for $P_i$ and $Q$. (Here, by $headVars(R)$ we denote the list of head variables of query $R$.) Recall that the head variables of $P_i^{exp}$ are the same as the head variables of

(CQ query) $P_i$, which we have defined as all the variables in the cross product defining $P_i$.

Now let $\mathcal{M}$ be the set of all mappings $\mu_{ij}$ from the relational subgoals of query $Q$ to *same-name* subgoals of $P_i^{exp}$; algorithm Build-MaxCR associates one set of databases $\mathcal{D}_{ij}$ with each $\mu_{ij} \in \mathcal{M}$. (Example 6 provides an illustration of mappings $\mu_{ij}$.) Fix an arbitrary mapping $\mu_{ij} \in \mathcal{M}$ and an arbitrary database $D \in \mathcal{D}_i$. The question that Build-MaxCR addresses at this point is "Does embedding the input query $Q$ in database $D$ using mapping $\mu_{ij}$ make the head of $Q$ true on $D$"? That is, when one composes the mapping $\mu_{ij}$ from $Q$ to $P_i^{exp}$ with the generative mapping $\iota$ from $P_i^{exp}$ to $D$, does the resulting mapping induce a (consistent) assignment mapping $\nu_{ij} : Q \rightarrow D$, with the property that $\nu_{ij}$ generates an answer to query $Q$ on database $D$.

Note that in case such an answer to $Q$ on $D$ exists, this answer is the tuple $\nu_{ij}(headVars(Q))$. Moreover, $\nu_{ij}$ happens to be the composition of two mappings: (i) $\tilde{\mu}_{ij}$, which is the mapping from the variables and constants of the query $Q$ to the variables and constants of $P_i^{exp}$, such that $\tilde{\mu}_{ij}$ is induced by $\mu_{ij}$, and (ii) the generative assignment mapping $\tilde{\iota}$ induced by the generative mapping $\iota$ from $P_i^{exp}$ to $D$.

Now we show the steps that algorithm Build-MaxCR takes to produce for each mapping $\mu_{ij}$ (i) its associated set $\mathcal{D}_{ij} \subseteq \mathcal{D}_i$, as well as (ii) the associated query $P'_{ij} = P_i \& S_{ij}$. This is the so-called *single-mapping processing* stage in the pseudocode of Algorithm 1.

For a fixed $P_i$ and for a fixed mapping $\mu_{ij} \in \mathcal{M}$, Build-MaxCR considers all the databases in the set $\mathcal{D}_i$ of databases that make the head of $P_i^{exp}$ true. Whenever, for a database $D \in \mathcal{D}_i$, embedding $Q$ in $D$ using $\mu_{ij}$ makes the head of $Q$ true on $D$, in the sense discussed above, algorithm Build-MaxCR classifies $D$ as belonging to the set of databases $\mathcal{D}_{ij}$ for $P_i$ and $\mu_{ij}$.

By construction, each database in the set $\mathcal{D}_{ij}$ makes the head of $P_i^{exp}$ true, via the generative assignment mapping $\tilde{\iota}$, and makes the head of $Q$ true, via the assignment mapping $\tilde{\iota} \circ \tilde{\mu}_{ij}$. Now Build-MaxCR uses each such $\mathcal{D}_{ij}$ to output, if possible, a single CQAC query $P'_{ij} = P_i \& S_{ij}$, such that $(P'_{ij})^{exp} \sqsubseteq Q$ and such that $S_{ij}$ is the logical OR of the total order of exactly the databases in $\mathcal{D}_{ij}$. Let $\bar{x}_{ij}$ be the list of variables (of $P_i$) defined as $\tilde{\mu}_{ij}(headVars(Q))$, where $\tilde{\mu}_{ij}$, a mapping on the variables and constants (as opposed to the relational subgoals) of $Q$, is induced by $\mu_{ij}$. We now fully define CQAC query $P'_{ij} = P_i \& S_{ij}$, by making $\bar{x}_{ij}$ its head variables.

Suppose all the variables that are used in $S_{ij}$ are variables of $P_i$ (or, in other words, are distinguished variables of $P_i^{exp}$); that is, $P'_{ij}$ is a safe query. In this case, Build-MaxCR outputs $P'_{ij}(\bar{x}_{ij})$. The basis for this decision is our proof of Theorem 3, where we show that $(P'_{ij})^{exp}$ is contained in $Q$. By this theorem, the containment is *by construction* of $P'_{ij}(\bar{x}_{ij})$ and thus does not require any containment test to be conducted by Build-MaxCR, once the safety condition is satisfied. That is, the only "containment test" that Build-MaxCR performs to ensure $(P'_{ij})^{exp} \sqsubseteq Q$ is the above safety test for $S_{ij}$ w.r.t. $P'_{ij}$; it is easy to see that the complexity of the safety test is linear in the size of $P'_{ij}$.

We conclude our discussion of how Build-MaxCR works by outlining the *multiple-mapping processing* stage (lines 15-18) in the pseudocode of Algorithm 1. This stage covers problem inputs for which MiniCon IP [11] cannot return a correct rewriting. Example 2 provides an illustration of a problem input requiring multiple-mapping processing. (Given the problem input of Example 2, Build-MaxCR returns correctly the rewriting $R$ of the example.) Essentially, even when some $P'_{ij}$'s (see line 13 of the pseudocode) fail the safety test in the single-mapping processing stage of Build-MaxCR, several of the $P'_{ij}$'s can be "put together", to ensure that in the resulting CQAC query, the summary ACs $S_{ij}$ do satisfy the safety condition. The algorithm does efficient enumeration of the sets $J$ outlined in the pseudocode, by using bitmaps that encode whether the answer to the input query $Q$ was empty or not "under" each mapping $\mu_{ij}$ on each database $D$ in the set $\mathcal{D}_i$ of databases that make the head of the query $P_i^{exp}$ true.

## 3.3 Examples illustrating Build-MaxCR

We now illustrate the work of Build-MaxCR using two examples. Example 6 illustrates the flow of the Build-MaxCR algorithm as described in Section 3.2 and in Algorithm 1 . We use Example 2 to show how Build-MaxCR finds a multiple-mapping rewriting, which would not have been discovered by an algorithm that considers only single-mapping rewritings.

### Single-Mapping Example

*Example 6.* Consider query $Q$ and views $U$, $V$, and $W$:

$Q(X)$ :- $p(X), s(X), X < 3$.
$U(A)$ :- $p(A), s(A)$.
$V(N)$ :- $p(N), N < 3$.
$W(L)$ :- $s(L)$.
$P'(A)$ :- $U(A), A < 3$.
$P''(N)$ :- $V(N), W(N)$.

In Example 6 suppose we are trying to find the $k$-bounded UCQAC MCR $P$ of $Q$ using $\mathcal{V} = \{U, V, W\}$ for $k = 2$. Initially, the outermost loop of Build-MaxCR sets $t = 1$, and $\mathcal{P}_1 = \{U(A), V(N), W(L)\}$. When $U(A)$ is the CQAC-rewriting template $P_i$, the set $MaxCrVars = VarsConsts(P_i^{exp}) \bigcup Consts(Q)$ is $MaxCrVars = \{A\} \bigcup \{3\} = \{A, 3\}$. Each possible total order on $MaxCrVars$ yields exactly one canonical database in the set $\mathcal{C}_i$ of MaxCR canonical databases for $P_i$ and $Q$. Thus, $\mathcal{C}_i = \{D_{A<3}, D_{A=3}, D_{A>3}\}$, where the total order $A < 3$ yields the MaxCR canonical database $D_{A<3}$, $A = 3$ yields $D_{A=3}$, and $A > 3$ yields $D_{A>3}$. Since the body of view $U$ does not contain any ACs, each database in $\mathcal{C}_i$ makes the head of $P_i^{exp}$ true. Hence the corresponding set $\mathcal{D}_i$ is the same as set $\mathcal{C}_i$, that is, $\mathcal{D}_i = \{D_{A<3}, D_{A=3}, D_{A>3}\}$. Build-MaxCR also determines the set $\mathcal{M}$ of all mappings from the relational subgoals of $Q$ to the same-name subgoals of $P_i^{exp}$. In this case, $\mathcal{M} = \{\mu\}$, where $\mu = \{p(X) \rightarrow p(A),\ s(X) \rightarrow s(A)\}$, and it induces $\tilde{\mu} = \{X \rightarrow A\}$. Next, Build-MaxCR produces $\mathcal{D}_{ij} \subseteq \mathcal{D}_i$ for each $\mu_j \in \mathcal{M}$. When $\mu_j$ is the $\mu$ in the example, $\mathcal{D}_{ij} = \{D_{A<3}\}$. This is because, for canonical database $D_{A<3}$, embedding $Q$ in $D_{A<3}$ using $\mu$ makes the head of $Q$ true, whereas for the remaining two canonical databases $D_{A=3}$ and $D_{A>3}$ from $\mathcal{D}_i$, embedding them in $Q$ using $\mu$ does not make the head of $Q$ true. The total order associated with $D_{A<3}$ is $A < 3$, and hence Build-MaxCR sets the set $S_{ij}$ of summary ACs to $\{A < 3\}$. (Note that if we had $|\mathcal{D}_{ij}| > 1$, then Build-MaxCR would have taken the logical OR of all total orders for $\mathcal{D}_{ij}$ to come up with $S_{ij}$, as illustrated in Section 3.3 in the description of Example 2.) Also Build-MaxCr sets $\bar{x}_{ij} = \tilde{\mu}_{ij}(headVars(Q)) = \mu(X) = A$. $S_{ij}$ contains only one variable $A$, and this variable is present in $P_i$. Since all variables of $S_{ij}$ also occur in $P_i$, the CQAC $P'_{ij}(\bar{x}_{ij})$ :- $P_i \& S_{ij}$, that is the CQAC $P'(A)$ :- $U(A), A < 3$, is added to the union $P$.

Note that for this definition of $P'$, we have $(P')^{exp} \sqsubseteq Q$ while $(P_i)^{exp} \not\sqsubseteq Q$. The difference is in the AC $A < 3$ that Build-MaxCR has conjoined with $P_i$ to get $P'$. In the AC $A < 3$, the presence of the constant 3 from $Q$ is an illustration of the need for the algorithm to use MaxCR canonical databases, as opposed to "standard" canonical databases for the CQAC-rewriting templates. Also, note that in Example 6, it turns out that $(P')^{exp} \equiv Q$. Build-MaxCR detects this and hence outputs $P'(A)$ :- $U(A), A < 3$ as its final answer and terminates all further processing. This is an illustration of Build-MaxCR terminating before the outermost $t$-loop goes all the way up to $k$. Since iterations for lower values of $t$ are faster than iterations for higher values of $t$, the main $t$-loop of MaxCR runs from 1 to $k$, rather than going in the other direction from $k$ down to 1.

Now suppose Example 6 is modified so that view $U$ is not available and we have with us only $\mathcal{V}' = \{V, W\}$. Then for the $t = 1$ iteration, Build-MaxCR sets $\mathcal{P}_1 = \{V(N), W(L)\}$. However, the body of $V$ does not have any $s$-subgoal, and the body of $W$ does not have any $p$-subgoal. So $\mathcal{M} = \emptyset$ in both cases, and so for $t = 1$, there is no rewriting that Build-MaxCR can add to $P$. Now for the $t = 2$ iteration, $\mathcal{P}_2 = \{P_1, P_2, P_3\}$, where $P_1$ is the CQAC-rewriting template $V(N_1), V(N_2)$, $P_2$ is $V(N), W(L)$, and $P_3$ is $W(L_1), W(L_2)$. Again, $\mathcal{M} = \emptyset$ for $P_1$ as well as $P_3$, so in these cases, there is no rewriting that Build-MaxCR can add to $P$. However, when $P_2$ is the CQAC-rewriting template $P_i$, $\mathcal{M} = \{\mu'\}$, where $\mu' = \{p(X) \rightarrow p(N), s(X) \rightarrow s(L)\}$, and it induces $\tilde{\mu}' = \{X \rightarrow N,\ X \rightarrow L\}$. The set $MaxCrVars = VarsConsts(P_i^{exp}) \bigcup Consts(Q)$ is $MaxCrVars = \{N, L\} \bigcup \{3\} = \{N, L, 3\}$. There are in all thirteen possible total orders on $\{N, L, 3\}$. Out of these there are exactly five total orders whose canonical databases make the head of $P_i^{exp}$ true. Hence, $\mathcal{D}_i$ is set to $\{D_{L<N<3}, D_{L=N<3}, D_{N<L<3}, D_{N<3=L}, D_{N<3<L}\}$. (Note that the body of view $V$ has the AC $N < 3$, and the remaining eight canonical databases from $\mathcal{C}_i$ that are not included in $\mathcal{D}_i$ are exactly those whose total orders have either $N = 3$ or $N > 3$. This illustrates how the presence of ACs in the body of $P_i^{exp}$ may prevent $P_i^{exp}$ from returning a nonempty answer on some databases in $\mathcal{C}_i$ and thus make $\mathcal{D}_i$ a proper subset of $\mathcal{C}_i$.) Furthermore, out of the five canonical databases in $\mathcal{D}_i$, there is only one canonical database, such that embedding $Q$ in that database using $\mu'$ makes the head of $Q$ true. Hence the set $\mathcal{D}_{ij}$ which corresponds to mapping $\mu_j = \mu'$ contains only that one canonical database $D_{L=N<3}$. Thus for $\mathcal{D}_{ij} = \{D_{L=N<3}\}$, Build-MaxCR finds the set of summary ACs $S_{ij} = \{L = N, L < 3, N < 3\}$. Since all variables of $S_{ij}$ also occur in $P_i$, the CQAC $P''_{ij}(\bar{x}_{ij})$ :- $P_i \& S_{ij}$, that is the CQAC $P''(N)$ :- $V(N), W(L), L = N, L < 3, N < 3$ with expansion $P''^{exp}(N)$ :- $p(N), s(L), L = N, L < 3, N < 3$, can be added to the union $P$. However, note that $L = N$ is an equality AC. Hence all occurrences of $L$ can be replaced by $N$, resulting in the CQAC $P''(N)$ :- $V(N), W(N)$ which is added to the union $P$.

**Multiple-Mapping Example**

In Example 2 suppose we are trying to find the $k$-bounded UCQAC MCR $P$ of $Q$ using $\mathcal{V} = \{V\}$ for $k = 1$. For the $t = 1$ iteration of the outermost loop, Build-MaxCR sets $\mathcal{P}_1 = \{V()\}$. When $V()$ is the CQAC-rewriting template $P_i$, the set $MaxCrVars = VarsConsts(P_i^{exp}) \bigcup Consts(Q)$ is $MaxCrVars = \{X, Y\} \bigcup \emptyset$ $= \{X, Y\}$. Each possible total order on $MaxCrVars$ yields exactly one canonical database in the set $\mathcal{C}_i$ of MaxCR canonical databases for $P_i$ and $Q$. Thus, $\mathcal{C}_i = \{D_{X<Y}, D_{X=Y}, D_{X>Y}\}$. Since the body of view $V$ does not contain any ACs, each database in $\mathcal{C}_i$ makes the head of $P_i^{exp}$ true. Hence the corresponding set $\mathcal{D}_i$ is the same as set $\mathcal{C}_i$, that is, $\mathcal{D}_i = \{D_{X<Y}, D_{X=Y}, D_{X>Y}\}$. Build-MaxCR also determines the set $\mathcal{M}$ of all mappings from the relational subgoals of $Q$ to the same-name subgoals of $P_i^{exp}$. In this case, $\mathcal{M} = \{\mu_1, \mu_2\}$, where $\mu_1 = \{p(A, B) \to p(X, Y)\}$ and $\mu_2 = \{p(A, B) \to p(Y, X)\}$. $\mu_1$ and $\mu_2$ induce $\tilde{\mu}_1 = \{A \to X, B \to Y\}$ and $\tilde{\mu}_2 = \{A \to Y, B \to X\}$, respectively. Next, Build-MaxCR produces $\mathcal{D}_{ij} \subseteq \mathcal{D}_i$ for each $\mu_j \in \mathcal{M}$. For $\mu_1$, $\mathcal{D}_{i1} = \{D_{X<Y}, D_{X=Y}\}$. This is because, for canonical databases $D_{X<Y}$ and $D_{X=Y}$, embedding $Q$ in them using $\mu_1$ makes the head of $Q$ true, but for the remaining canonical database $D_{X>Y}$ from $\mathcal{D}_i$, embedding $D_{X>Y}$ in $Q$ using $\mu_1$ does not make the head of $Q$ true. Similarly, Build-MaxCR also constructs $\mathcal{D}_{i2} = \{D_{X>Y}, D_{X=Y}\}$ for $\mu_2$. The total order associated with $D_{X<Y}$ is $X < Y$, and the total order associated with $D_{X=Y}$ is $X = Y$. Hence for $j = 1$, Build-MaxCR determines the set $S_{ij}$ (of summary ACs that characterize exactly the canonical databases in $\mathcal{D}_{ij}$) as $S_{i1} = \{X \leq Y\}$. Thus for the $j = 1$ case, this is an illustration of Build-MaxCR finding the set $S_{ij}$ by taking the logical OR of the total order ACs for the set $\mathcal{D}_{ij}$. Similarly, for $j = 2$, since $\mathcal{D}_{i2} = \{D_{X>Y}, D_{X=Y}\}$, Build-MaxCR determines $S_{i2} = \{X \geq Y\}$ by taking the logical OR of $X > Y$ and $X = Y$ that are associated with $D_{X>Y}$ and $D_{X=Y}$, respectively. Note that in this example, for $j = 1$ and for $j = 2$, $S_{ij}$ contains variables $X$ and $Y$, which do not occur in $P_i$. Thus $P_i \& S_{i1}$ cannot be used to form a safe query $P'_{i1}$ which can be added to the union $P$. Similarly $P_i \& S_{i2}$ too cannot be used to form a safe query $P'_{i2}$ which can be added to $P$. However, Build-MaxCR finds set $J = \{1, 2\}$, $|J| > 1$, for which $\bar{x}_{i1} = \mu_1(headVars(Q))$ and $\bar{x}_{i2} = \mu_2(headVars(Q))$ are the same. This is an illustration of the multiple-mapping processing done by Build-

MaxCR. For this set $J$, Build-MaxCR finds $\tilde{D}_i = \mathcal{D}_{i1} \bigcup \mathcal{D}_{i2} = \{D_{X<Y}, D_{X=Y}\} \bigcup \{D_{X>Y}, D_{X=Y}\} = \{D_{X<Y}, D_{X=Y}, D_{X>Y}\}$, and $\tilde{S}_i = \emptyset$ is the corresponding set of summary ACs for $\tilde{D}_i$. Now, for this $\tilde{S}_i$, it is true (trivially, since $\tilde{S}_i$ has no variables) that all variables used in $\tilde{S}_i$ also occur in $P_i = V()$. Hence Build-MaxCR adds the CQAC $P'_i()$ :- $P_i \& \tilde{S}_i$, that is, the CQAC $P'()$ :- $V()$ to the union $P$. This completes the Build-MaxCR processing for $k = 1$. $\square$

In the rest of this Section, we formulate theorems that establish the soundness and completeness of Build-MaxCR. In Section 3.4 we develop the theory of total-order CQAC queries. This is a contribution of independent interest to any work that considers all canonical databases of a CQAC query. In our case, this contribution has direct application in developing the proofs of soundness and completeness of Build-MaxCR, which are presented in Section 3.5 and Section 3.6, respectively.

## 3.4 Theoretical Results on Total-Order CQAC Queries

Note that the Build-MaxCR algorithm does not have to do containment checking, either during the construction of the outputs or (as final containment test) to test that the MaxCR outputs are contained in the input query.

In the formal statements below, for a Build-MaxCR problem input $(Q, \mathcal{V}, k)$, $P$ denotes a CQAC-rewriting template for $Q$, of some size $t \leq k$. When we consider $P$ as a query (rather than just as a cross product), we define $P$ as a CQ query without nondistinguished variables.

**Proposition 2.** *Let $D$ be a MaxCR canonical database for $P$ and $Q$, such that $S$ is the total order for $D$. Then under set semantics, query $P'\ :-\ P^{exp} \& S$ has exactly one tuple $t$ in the answer on $D$,* regardless of the choice of head variables of $P'$.[11] *Further, $t$ can be obtained by applying to the head variables of $P$ the generative assignment mapping (see Definition 11) from $P^{exp}$ to $D$.*

*Proof.* On the $n$ variables of $P'$, total order $S$ enforces a fixed number of $m \leq n$ distinct values. The database $D$, which is associated with $S$, by construction also has $m$ distinct stored values. That means that the association between the variables of $P'$ and the stored values of $D$ is 1:1. Let $M$ be the total-order mapping associated with $D$ and $S$. Observe that $M$, when treated as an assignment mapping from the relational subgoals of $P'$ (i.e., of $P^{exp}$) to $D$, (1) provides exactly this 1:1 association, and (2) produces an answer tuple to $P'$ on $D$. Recall that by Definition 11 $M$ is indeed the generative assignment mapping from $P^{exp}$ to $D$.

**Proposition 3.** *Let $D$ be a MaxCR canonical database for $P$ and $Q$, such that $S$ is the total order for $D$. Let $S$ enforce $m$ distinct values on the contents of $D$. Denote by $P'$ the CQAC query whose body is $P^{exp} \& S'$, where $S'$ is a total order on a MaxCR database for $P$ and $Q$, and whose head variables are* an arbitrary

[11] Note that by construction of MaxCR canonical databases, queries $P'$ are safe in the context of Propositions 2 through 5.

subset of the body variables of $P$.[12] *Then query $P'$ has an empty answer on $D$ whenever $S'$ enforces $m' > m$ distinct values on the variables of $P'$.*

The proof of Proposition 3 is an immediate variation on the observations in the proof of Proposition 2.

Note that when $m' < m$, in the terminology of Proposition 3, and even when $m' = m$ (while $S'$ is still not the same as $S$), then it is also possible that we get an empty answer to query $P'$ on database $D$. At the same time, in some cases we *can* get one or even more than one nonempty answers to $P'$.

Here are two examples that illustrate this point. In Example 7, the total order that defines $P'$ enforces the same number of distinct values as the total order on the given MaxCR canonical database $D$. In this case, even though the total orders of $P'$ and $D$ are not the same, the answer to $P'$ on $D$ is not empty. Example 8, on the other hand, shows that *more than one* answer tuples to $P'$ can be obtained on a MaxCR canonical database $D$, in case where the total order of $D$ enforces strictly more distinct values than the total order defining $P'$.

*Example 7.* Consider a CQAC-rewriting template $P$ defined in terms of a single view $V$:[13]

$P(X, Y) \ :- \ V(X, Y).$
$V(X, Y) \ :- \ s(X, Y), s(Y, X).$
$P^{exp}(X, Y) \ :- \ s(X, Y), s(Y, X).$

Let the total order $S'$ be $S' : X < Y$, and define $P'$ as

$P'(X, Y) \ :- \ s(X, Y), s(Y, X), X < Y.$

That is, the body of $P'$ is $P^{exp} \& (X < Y)$.

Let $D$ be the MaxCR canonical database for $P$ and for the input query $Q$, such that $D$ is associated with total order $S : Y < X$. (Note that $S$ and $S'$ are not the same.) Suppose $D = \{s(1, 2), s(2, 1)\}$. (When constructing $D$, we arbitrarily assigned the stored values as $1 = Y < X = 2$.) Then the answer $(1, 2)$ to query $P'$ on $D$ can be obtained by using the assignment mapping from $P^{exp}$ to $D$ that maps $X$ into 1 and $Y$ into 2.

*Example 8.* Consider a CQAC-rewriting template $P$ defined in terms of a single view $V$:

$P(X, Y, Z, T) \ :- \ V(X, Y, Z, T).$
$V(X, Y, Z, T) \ :- \ s(X, Y), s(Y, Z), s(Z, T).$
$P^{exp}(X, Y, Z, T) \ :- \ s(X, Y), s(Y, Z), s(Z, T).$

[12] Recall that all variables of both $P$ and $P^{exp}$ are distinguished, and that the sets of body variables of $P$ and $P^{exp}$ are the same.

[13] In all the examples in this section, we assume that the query $Q$, in the Build-MaxCR problem input, does not use constants. As a consequence, each MaxCR canonical database for $P$ and $Q$ is a (standard) canonical database for $P^{exp}$. Thus the definition of $Q$ is, in a sense, irrelevant to the examples in this section.

Let the total order $S'$ be $X = Y = Z = T$, and define $P'$ as

$$P'(X, Y, Z, T) \; :- \; s(X, Y), s(Y, Z), s(Z, T), X = Y = Z = T.$$

That is, the body of $P'$ is $P^{exp}\&(X = Y = Z = T)$.

Let $D$ be the MaxCR canonical database for $P$ and for the input query $Q$, such that $D$ is associated with total order $S : X = Y < Z = T$. (Note that $S$ and $S'$ are not the same.) Suppose $D = \{s(1, 1), s(1, 2), s(2, 2)\}$. (When constructing $D$, we arbitrarily assigned the stored values as $1 = X = Y < Z = T = 2$.) Then there are *two* answers, $(1, 1, 1, 1)$ and $(2, 2, 2, 2)$, to query $P'$ on $D$.

**Proposition 4.** *Let $D$ be a MaxCR canonical database for $P$ and $Q$, such that $S$ is the total order for $D$. Let $S$ enforce $m$ distinct values on the contents of $D$. Denote by $P'$ the CQAC query whose body is $P^{exp}\&S'$, where $S'$ is a total order on a MaxCR database for $P$ and $Q$, and whose head variables are* an arbitrary subset of the body variables of $P$. *Then under set semantics query $P'$ has at most one answer on $D$ whenever the total order $S'$ of $P'$ enforces $m' = m$ distinct values on the variables of $P'$.*

*Proof.* There are two cases:

Case 1 is $S = S'$; the claim of Proposition 4 for this case holds by Proposition 2.

Case 2 is $S \neq S'$. In this case, suppose the answer to $P'$ on $D$ is not empty. Then we use the reasoning in the proof of Proposition 2 to argue that there is a 1:1 association between all the body variables of $P'$ and the stored values in $D$, thus $P'$ has exactly one answer tuple, say $t$, on $D$ under set semantics.

Note that, unlike Case 1 of this proof, $t$ cannot be obtained by applying the generative assignment mapping from the body of $P'$ to the stored values in the database $D$. Instead, consider an isomorphism $\nu$ on the set of body variables of $P'$, such that $\nu$ maps the smallest-value variable w.r.t. total order $S'$ to the smallest-value variable w.r.t. total order $S$, and so on, in the increasing order of variable values according to the two total orders. *Note that $\nu$ is a 1:1 mapping.* (It is easy to construct counterexamples to the existence of the answer to $P'$ on $D$ whenever $m' = m$ but $\nu$ is not a 1:1 mapping.)

Then the assignment mapping that generates the tuple $t$ is the composition of the isomorphism $\nu$ with the total-order mapping $M$ for the database $D$. For instance, in Example 7, the assignment mapping that generates tuple $(1, 2)$ is a composition of $\nu = \{X \rightarrow Y, Y \rightarrow X\}$ with $M$ (for $D$ and $S$) where $M = \{Y \rightarrow 1, X \rightarrow 2\}$.

**Proposition 5.** *Let $D$ be a MaxCR canonical database for $P$ and $Q$, such that $S$ is the total order for $D$. Let $S$ enforce $m$ distinct values on the contents of $D$. Denote by $P'$ the CQAC query whose body is $P^{exp}\&S'$, where $S'$ is a total order on a MaxCR database for $P$ and $Q$, and whose head variables are* an arbitrary subset of the body variables of $P$. *Suppose the total order $S'$ of $P'$ enforces $m' < m$ distinct values on the variables of $P'$, and suppose the answer to query $P'$ on $D$ includes tuple $t$. Then there exists a MaxCR canonical database $D'$ for $P$ and $Q$, such that:*

1. *$D'$ can be obtained by using a subset of the tuples in $D$, and*
2. *the answer $t$ to $P'$ on $D$ can be obtained by applying the generative mapping from $P'$ to $D'$.*

*Proof.* From the existence of tuple $t$ in the answer to $P'$ on $D$, there exists an assignment mapping $\tilde{\lambda}$ from $P'$ to $D$, such that $t$ can be obtained by using $\tilde{\lambda}$. Let $\lambda$ be the mapping from the relational subgoals of $P'$ to the stored tuples of $D$, such that $\lambda$ is induced by $\tilde{\lambda}$. Consider the set $T$ of those stored tuples of $D$ that are in the image of all the relational subgoals of $P'$ under $\lambda$. Because $S'$ is a total order on $P'$, from the existence of tuple $t$ it follows that (i) the tuples in $T$ have a total of $m'$ distinct values, and (ii) there is a 1:1 mapping, which happens to be $\tilde{\lambda}$, between the variables of $P'$ and the $m'$ distinct values in the tuples of $T$. (See proof of Proposition 2 for the details of this reasoning.)

We now prove Claims 1 and 2 of the Proposition, by showing that the set $T$ can be used to form a MaxCR canonical database $D'$ for $P$ and $Q$. Indeed, it is easy to see that $\tilde{\lambda}$ is a generative assignment mapping from $P'$ to the database $T$.

**Corollary 1.** *In the setting of Proposition 5, the answer to $(P')^{exp}$ on $D'$ has exactly one answer tuple, under set semantics for query evaluation.*

The claim of Corollary 1 follows immediately from Proposition 2 and from the construction of $D'$ as outlined in the proof of Proposition 5.

**Proposition 6.** *Let $D$ be a MaxCR canonical database for $P$ and $Q$, such that $S$ is the total order for $D$. Let $S$ enforce $m$ distinct values on the contents of $D$. Let the total order $S' \neq S$ of $P' = P^{exp} \& S'$ enforce $m' = m$ distinct values on the variables of $P'$. Then, whenever $P'$ produces an answer tuple $t$ on $D$, then $Q$ will also produce $t$ on $D$.*

*Proof.* Let $\tilde{\lambda}$ be the assignment mapping that produces the answer $t$ to $P'$ on the database $D$. From $m = m'$, it follows that $\tilde{\lambda}$ enforces a 1:1 mapping between the variables/constants of $P'$ and the values stored in $D$. It follows that we can interpret $\tilde{\lambda}$ as a "generative" mapping from $P'$ to $D$, and thus can interpret $D$ as "the native" MaxCR canonical database for $P'$ (and $Q$). (That is, we can interpret the total order on $D$ as *the* total order for $P'$, by observing that $\tilde{\lambda}(S')$ is a total order on the values stored in $D$.)

Now suppose, toward contradiction, that this generative assignment mapping does not produce an answer tuple to $P'$ on $D$. (If such an answer is produced, then it is clear that the answer is exactly $t$.) But it is clear that $\tilde{\lambda}$ produces $t$ when we use the "original" (i.e., "non-native canonical database") interpretation of $D$ w.r.t. $P'$. Hence the contradiction.

By construction of Build-MaxCR, $Q$ produces, on each MaxCR canonical database $D$, whatever answer tuple $t$ is produced by the "native" $P'$ for $D$. Q.E.D.

## 3.5 Proof of Soundness of Algorithm Build-MaxCR

**Theorem 3. (Soundness of Build-MaxCR)** *For a Build-MaxCR problem input $(Q, \mathcal{V}, k)$, let $P$ be a CQAC-rewriting template (of some size $s \leq k$). Then for any CQAC query $P' \; : - \; P \& S$ that is output by Build-MaxCR, $(P')^{exp}$ is contained in $Q$.*

Given input $(Q, \mathcal{V}, k)$ to Build-MaxCR and CQAC-rewriting template $P$ for this input, let $P'$ be a CQAC query output by Build-MaxCR. By Lemma 1, $(P')^{exp}$ is equivalent to the union $\bigcup_{all \; i} P'_i$, where each $P'_i$ has the same head arguments as $P'$ and has the body $P^{exp} \& S_i$, for some total order $S_i$ that implies the summary ACs $S$ of $P'$. Then it is clear that proving $P'_i \sqsubseteq Q$, for all $i$, proves the claim of the theorem. Thus, in the remainder of the proof we show $P'_i \sqsubseteq Q$ for an arbitrary $P'_i$ that satisfies the above conditions.

The idea of the proof is to show that on all MaxCR canonical databases $D$ for $P$ and $Q$ (and thus on all *canonical* – in the original sense of the term, see Definition 5 – databases for $P'_i$), the answer to $Q$ on $D$ is a superset of the answer to $P'_i$ on $D$. This proves $P'_i \sqsubseteq Q$ by the canonical-database containment test.

1. Suppose the total order $S_i$ for $P'_i$ enforces $m_i$ distinct values on the variables of $P'_i$.
2. Fix a MaxCR canonical database $D$ for $P$ and $Q$, such that $S$ is the total order for $D$. Let $S$ enforce $m$ distinct values on $D$.
   There are four cases:
   (a) $m = m_i$ and $S = S_i$; then, by Proposition 2 and by construction of the answer to $Q$ on $D$, $Q$ produces the (only) answer tuple to $P'_i$ on $D$; here we use the fact that Build-MaxCR tests the nonemptiness of the answer to $Q$ on $D$ by using the composition of (i) $\tilde{\mu}_{ij}$ for $Q$ and $P'_i$, and of (ii) the generative mapping $\tilde{\iota}$ from $P'_i$ to $D$; note that the relational subgoals of $P^{exp}$ and of $P'_i$ are the same, thus both $\tilde{\mu}_{ij}$ and $\tilde{\iota}$ "make sense" for $P'_i$ and $Q$)
   (b) $m = m_i$ and $S \neq S_i$; suppose that the answer to $P'_i$ on $D$ is not empty (otherwise the containment $P'_i \sqsubseteq Q$ is obvious);
   then it follows from Proposition 4 that we can treat $D$ as a "permutation" of the MaxCR canonical database for $P'_i$ and $Q$, such that the total order for $D$ is $S_i$ (i.e., the same as the total order for $P'_i$);
   let $\tilde{\lambda}$ be the assignment mapping from $P'_i$ to $D$ such that $\tilde{\lambda}$ produces an answer tuple $t$ to $P'_i$ on $D$; then from the proof of Proposition 4 it follows that (i) $t$ is the only answer to $P'_i$ on $D$, and (ii) $\tilde{\lambda}$ can be treated as the generative mapping from $P'_i$ to $D$, hence this case reduces to (a) above;
   (c) $m < m_i$; in this case, by Proposition 3, the answer to $P'_i$ on $D$ is empty;
   (d) $m > m_i$; then, by Proposition 5, each tuple in the answer to $P'_i$ on $D$ is obtained using an assignment mapping from $P'_i$ into a substructure $D'$ of $D$ such that $D'$ is a MaxCR canonical database for $P'_i$ and $Q$ with a total order that is the same as the total order for $P'_i$;
   thus, from Proposition 5 and from Corollary 1 it follows that each such case is the same as case $m = m_i$ and $S = S_i$ above.

3. In all the four cases, we have proved that $Q$ produces all answers to $P_i'$ on $D$. There are no other possibilities for the relationship between $P_i'$ and MaxCR canonical databases for $P_i'$ and $Q$, thus Q.E.D.

**Lemma 1.** *Given input $(Q, \mathcal{V}, k)$ to Build-MaxCR and CQAC-rewriting template $P$ for this input. Let $P' = P\&S$, where $S$ is a set of ACs on the head variables of $P$, be a CQAC query output by Build-MaxCR based on $P$. Let $S_1, S_2, \ldots, S_m$ be all ways of expanding $S$ to total orders on the set $V$ that comprises (i) all variables and constants of $P^{exp}$, and (ii) all constants of $Q$. Then $(P')^{exp} \equiv \bigcup_{i=1}^{m} P^{exp}\&S_i$.*

The proof of the lemma is immediate by construction of $S$ in algorithm Build-MaxCR.

### 3.6 Proof of Completeness of Algorithm Build-MaxCR

**Theorem 4. (Completeness of Build-MaxCR)** *For a Build-MaxCR problem input $(Q, \mathcal{V}, k)$, let $\mathbf{R}$ be a UCQAC query defined in terms of $\mathcal{V}$, such that (i) in each CQAC component $R_i$ of $\mathbf{R}$, the number of relational subgoals of $R_i$ does not exceed $k$, and (ii) $\mathbf{R}^{exp} \sqsubseteq Q$. Then (1) the output of Build-MaxCR is not empty, and (2) denoting by $\mathbf{P}$ the UCQAC output of Build-MaxCR, we have that $\mathbf{R}^{exp} \sqsubseteq \mathbf{P}^{exp}$.*

*Proof.* Let $\mathcal{D}$ be the schema used for defining the query $Q$ and all the views in $\mathcal{V}$. Consider an arbitrary database $D$ with schema $\mathcal{D}$, such that the answer to $\mathbf{R}^{exp}$ on $D$ is not empty. Suppose $t$ is a tuple in the answer to $\mathbf{R}^{exp}$ on $D$. To prove this completeness theorem, it is enough to show that there exists a CQAC $P_r$ that is defined in terms of $\mathcal{V}$, such that on input $(Q, \mathcal{V}, k)$, $P_r$ has been output by Build-MaxCR,[14] and such that $t \in P_r^{exp}(D)$.

We "expand" the UCQAC $\mathbf{R}$ into its CQAC components, as $\mathbf{R} = \cup_{i=1}^{m} R_i$ for some natural number $m$. Here, each $R_i$ is a CQAC component of $\mathbf{R}$. By $t$ being a tuple in the answer to $\mathbf{R}^{exp}$ on $D$, there exists a CQAC $R_i$ in $\mathbf{R}$, such that $t \in R_i^{exp}(D)$. (In case more than one CQAC in $\mathbf{R}^{exp}$ returns $t$ on $D$, we choose an arbitrary such CQAC $R_i^{exp}$.) We represent $R_i^{exp}$ as a union of total-order CQACs where each total order is on all the variables and constants of $R_i^{exp}$: $R_i^{exp} = \cup_{j=1}^{l} R^*\&S_j$, for some natural number $l$. Here, each $R^*\&S_j$ is a CQAC in the normal form, that is $S_j$ is a total order on all the variables and constants of $R_i^{exp}$, and $R^*$ is a cross product of all the relational subgoals of $R_i^{exp}$, such that no variable name occurs in $R^*$ twice and such that $R^*$ contains no constants. Observe that $R^*$ is the relational part of (the normal forms of) *all* the CQACs in the union $R_i^{exp} = \cup_{j=1}^{l} R^*\&S_j$.

By $t \in R_i^{exp}(D)$, there must exist an $S_j$ (for some $j \in \{1, \ldots, l\}$) such that $t \in (R^*\&S_j)(D)$. We denote $R^*\&S_j$, for this fixed $j$, by $R_j^*$.

Let $\lambda$ be an assignment mapping that produces the answer $t$ to $R_j^*$ on the database $D$. By definition, $\lambda$ satisfies the conjunction of relational atoms $R^*$

[14] Thus $P_r$ is guaranteed to have at most $k$ relational subgoals.

w.r.t. $D$, and applying $\lambda$ to $S_j$ produces a true conjunction of arithmetic comparisons on constants. Note that $\lambda$ is a homomorphism from $R^*$ to the stored tuples in the database $D$. (To recast an assignment mapping as a homomorphism, we use an equivalent representation of stored tuples of a database as ground relational atoms.) Now let $T$ be the database (with schema $\mathcal{D}$) containing exactly the tuples in $D$ that are images of all the relational atoms in $R^*$ under $\lambda$. In the remainder of the proof, we show that Build-MaxCR has considered (a database isomorphic to) database $T$ when processing $R^*$, and that the algorithm output a CQAC $P_r$ such that $t \in P_r^{exp}(D)$.

Observe that by construction of Build-MaxCR, the algorithm has considered $R^*$ when processing input $(Q, \mathcal{V}, k)$. Indeed, $R^*$ conjoined with some portion $S'_j$ of the ACs in $S_j$ is, by definition of $R_i^{exp}$, an expansion of a cross product, call it $\mathbf{V}_{(n)}$, of some $n \leq k$ subgoals, where the predicate for each subgoal corresponds to a view name in $\mathcal{V}$. Thus, Build-MaxCR has considered both $R^* \& S'_j$ and all MaxCR canonical databases for $R^* \& S'_j$.

In the remainder of the proof, we will use the following observation. In the $S_j$ for our fixed $j$, all the ACs in the portion $S''_j = S_j - S'_j$, with "$-$" understood as set difference, are either comparisons of variables of $R^*$ with those constants that do not occur in the expansion of $\mathbf{V}_{(n)}$, or (some of the) ACs that impose the total order on the variables and constants in $R_i^{exp}$. That is, $S''_j$ does not contain any ACs that are implied by the ACs in the expansion of $\mathbf{V}_{(n)}$. This observation follows from our assumption that all (U)CQAC queries that we consider in this paper are safe.

We now show that Build-MaxCR has considered (a database isomorphic to) database $T$ when processing $R^*$. We proceed in two steps:

(1) For each constant $c$ occurring in $S''_j$ but not in $S'_j$, drop from $S_j$ all the ACs containing $c$. Denote the result of this AC removal by $S_j^{(1)}$. Observe that the queries $R^* \& S_j$ and $R^* \& S_j^{(1)}$ return the same set of answers on the database $T$. (This follows from our results of Section 3.4 on total-order CQAC queries.) We assume here that the heads of $R^* \& S_j$ and of $R^* \& S_j^{(1)}$ are the same. We denote by $U^{(1)}$ the set of all variables and constants occurring in $R^* \& S_j^{(1)}$; note that $S_j^{(1)}$ is a total order on $U^{(1)}$.

(2) Let $C_Q$ be the set of all constants that occur in the query $Q$ but not in $U^{(1)}$. We use $C_Q$ to add to $S_j^{(1)}$ all ACs that are necessary to obtain a total order on $U^{(1)} \cup C_Q$. Denote by $S_j^{(2)}$ the resulting set of total-order ACs. Again, by our results of Section 3.4 on total-order CQAC queries, all of $R^* \& S_j$, $R^* \& S_j^{(1)}$, and $R^* \& S_j^{(2)}$ return the same set of answers on the database $T$. We assume here that the heads of $R^* \& S_j^{(1)}$ and of $R^* \& S_j^{(2)}$ are the same.

By construction of $S_j^{(2)}$ it holds that $T$ is a *MaxCR database* for (the input query $Q$ and) $R^* \& S'_j$, such that $S_j^{(2)}$ is the total order associated with $T$. We have seen that $R^* \& S_j^{(2)}$ returns on $T$ the same set of answers as a CQAC element $R^* \& S_j$ of $\mathbf{R}^{exp}$, and therefore returns on $T$ the same set of answers as the query

$Q$. Thus, by construction of Build-MaxCR, Build-MaxCR must have considered $R^* \& S_j^{(2)}$ when processing input $(Q, \mathcal{V}, k)$.

We have seen that on each database $D$ on which $\mathbf{R}^{exp}$ produces an answer,[15] expansions of some CQACs considered by Build-MaxCR also produce exactly the answers to $\mathbf{R}^{exp}$(D). Thus, to show containment of $\mathbf{R}^{exp}$ in the expansion of the output of Build-MaxCR for the input $(Q, \mathcal{V}, k)$, it remains to show that for all the (considered above) CQACs $P_r$ that "cover" $\mathbf{R}^{exp}$ and that have been considered by Build-MaxCR, the algorithm returns each such $P_r$.

Indeed, the only case where Build-MaxCR would not return a $P_r$ in question would be the case where the summary ACs of $P_r$ (obtained by Build-MaxCR from the ACs of the individual total-order CQAC components $P'_r$ of $P_r^{exp}$) would not be enforceable on just the head variables of $P_r$. We observe first that for each such $P'_r$, the head variables of $P'_r$ would be the same as the head variables of some CQAC in $\mathbf{R}$. (This follows from the construction of the $R^* \& S_j^{(2)}$, see earlier in this proof.)

Now, for some $P_r$ considered by Build-MaxCR for $(Q, \mathcal{V}, k)$, let $c$ be a constant of the query $Q$ such that $c$ does not occur in the expansion of the $\mathbf{V}_{(n)}$ used to build $P_r$. (That is, $P_r$ is defined as a conjunction of $\mathbf{V}_{(n)}$ with some ACs.) Assume that Build-MaxCR is not returning this $P_r$ because an AC $S(c)$ that uses the constant $c$ involves a *nonhead* variable of $P_r$. In this case, it must be that $\mathbf{R}^{exp}$ is not contained in $Q$, because the only case in which $S(c)$ would arise in $P_r$ is the case where one must "remove from the output" of Build-MaxCR a MaxCR CQAC $P'_r$ that returns a nonempty answer on some database on which $Q$ does not return any answer. Thus, we obtain by contradiction that our assumption (of Build-MaxCR not being able to produce an output due to $S(c)$ not being enforceable on the head variables of $P_r$) cannot hold.

From the proof in the preceding paragraph, it holds that for each CQAC $R_i$ in $\mathbf{R}$, there must exist a CQAC $P_j$ in the output $\mathbf{P}$ of Build-MaxCR (that is, $\mathbf{P} = \cup_{j=1}^{z} P_j$ for some natural number $z$) such that (1) the relational parts of $R_i$ and $P_j$ are isomorphic (after dropping any duplicate subgoals), and (2) the AC part of $R_i$ implies the AC part of $P_j$. Therefore, we have shown $\mathbf{R}^{exp} \sqsubseteq \mathbf{P}^{exp}$ as required.

## 4 Minimally Containing Rewritings

We now turn to the problem of finding minimally containing rewritings [18–20], which we abbreviate as *MiCRs,* of a CQAC query using CQAC views. The word "minimal" in "MiCR" refers to a containing rewriting that contains the fewest false positives (in the given rewriting language) w. r. t. the query answer.

We focus on the problem of enabling a MiCR of a CQAC query using CQAC views to be executed as efficiently as possible. To that end, we look at minimizing the number of relational subgoals of a given MiCR, and thus the number of joins in the evaluation plans for the MiCR. In Section 4.1, we introduce the

[15] By $\mathbf{R}^{exp} \sqsubseteq Q$, all answers produced by $\mathbf{R}^{exp}$ on $D$ are also produced by $Q$ on $D$.

notion of a *minimized MiCR,* which formalizes the above efficiency intuition. The main contribution of this section is an algorithm that we call *pruned-MiCR,* see Section 4.3. Given a CQAC MiCR for a given problem input (i.e., for a CQAC query and a set of CQAC views), pruned-MiCR *globally* minimizes the MiCR in an *efficient* and *scalable* way. (See Section 4.4 for the correctness and complexity results for pruned-MiCR.) Our experimental results in Section 6 suggest that for many problem inputs (for the MiCRs for queries and views of certain types), pruned-MiCR outputs minimized MiCRs whose evaluation costs are significantly lower than those of the (MiCR) input to the algorithm.

Note that the idea of pruned-MiCR is quite general and thus applicable beyond containing rewritings. Specifically, a straightforward modification of pruned-MiCR could be used to reduce the number of relational subgoals of (and thus to provide more efficient execution options for) the outputs of our Build-MaxCR algorithm of Section 3. Suppose the Build-MaxCR algorithm has been executed to obtain the MCR of query $Q$ using the viewset $\mathcal{V}$. Then each CQAC component, say $R$, in the UCQAC output of the Build-MaxCR algorithm can be minimized by running the pruned-MiCR algorithm on input $(Q, \mathcal{V}, R)$, and then replacing the CQAC $R$ in the Build-MaxCR output, by the answer $R'$ output by the pruned-MiCR algorithm. When this has been done for each $R$ in the output of Build-MaxCR, the resulting UCQAC MCR is equivalent (as expansions) to the original UCQAC MCR that was output by Build-MaxCR. Yet, the new UCQAC is likely to be much faster to execute. This could be especially important in applications where the speed of execution of the rewriting is much more critical than the speed of generation of the rewriting, for example in applications where the rewriting is obtained once, and then executed several times.

## 4.1 The Setting and Definitions

We begin by providing a general definition of a MiCR and by defining (CQAC) minimized MiCRs.

**Definition 12. (Minimally containing rewriting)** *A query $Q'$ defined in query language $\mathcal{L}_1$ is a* minimally containing rewriting (MiCR) *of a query $Q$ defined in language $\mathcal{L}_2$ using a set of views $\mathcal{V}$ defined in language $\mathcal{L}_3$ if: (1) $Q'$ is a containing rewriting of $Q$ in terms of $\mathcal{V}$, and (2) there exists no containing rewriting (in language $\mathcal{L}_1$) $Q''$ of $Q$ using $\mathcal{V}$, such that the expansion of $Q''$ is properly contained in the expansion of $Q'$.*

For the results in this section, each of $\mathcal{L}_1$ through $\mathcal{L}_3$ is the language of CQAC queries.

We study the problem of minimizing the number of relational subgoals of a given CQAC MiCR, to enable efficient evaluation of the MiCR. We now define the notion of *minimized MiCR,* which formalizes this efficiency intuition.

**Definition 13. (Minimized MiCR)** *Given a CQAC query $Q$ and a set of CQAC views $\mathcal{V}$, CQAC MiCR $R$ of $Q$ using $\mathcal{V}$ is a* minimized (CQAC) MiCR *of $Q$ using $\mathcal{V}$ if removing any relational subgoal of $R$ results in query $R'$ such that $R$ and $R'$ are not equivalent as expansions, that is $R^{exp} \not\equiv (R')^{exp}$.*

By definition, if we delete even a single relational subgoal from a minimized MiCR, it no longer remains a MiCR. Finding minimized MiCRs is especially important in those cases where the MiCR is computed once and then executed repeatedly. In such cases, it is important that the MiCR execute efficiently. Since a minimized MiCR may have many fewer relational subgoals than the original MiCR (see, e.g., Example 11), and thus many fewer joins, such a performance improvement would have a significant payoff.

We now consider the notion of a "globally minimal" minimized MiCR. A *globally minimal minimized CQAC MiCR* for a CQAC query $Q$ and set $\mathcal{V}$ of CQAC views has the minimum number of relational subgoals among all CQAC queries defined using $\mathcal{V}$ that are equivalent (as expansions) to a (unique) CQAC MiCR for $Q$ and $\mathcal{V}$. It turns out that a globally minimized MiCR may not be unique for a given $(Q, \mathcal{V})$, as shown by the following example.

*Example 9.* Consider a Boolean CQ query $Q$ and two Boolean CQ views $V_1$ and $V_2$. (Recall that CQ queries are in the language CQAC.)

$Q() \ :- \ p(X).$
$V_1() \ :- \ p(X).$
$V_2() \ :- \ p(X).$

Any sound and complete algorithm for generating CQAC MiCRs for CQAC inputs would return rewriting

$R() \ :- \ V_1(), V_2().$

This MiCR $R$ (for $Q$ and $\{V_1, V_2\}$) is equivalent as expansionsto each of the queries $R_1$ and $R_2$, as follows:

$R_1() \ :- \ V_1().$
$R_2() \ :- \ V_2().$

Each of $R_1$ and $R_2$ is a globally minimal minimized MiCR for $(Q, \{V_1, V_2\})$. $\square$

Next, we give an example which shows that two distinct minimized MiCRs for a given CQAC MiCR can have a different number of relational (view) subgoals. At the same time, note that the minimized MiCRs output by our algorithm pruned-MiCR is guaranteed to be a *globally* minimized MiCR, see Section 4.4 for the details.

*Example 10.*
$Q() :- p(), s(), t(), u().$
$V_1() :- p(), s().$
$V_2() :- t(), u().$
$V_3() :- s(), t().$
$V_4() :- p().$
$V_5() :- u().$

$R() :- V_1(), V_2(), V_3(), V_4(), V_5().$

$R'() : -V_1(), V_2().$
$R''() : -V_3(), V_4(), V_5().$

In this example $R$ is the full MiCR of $Q$ using views $V_1, V_2, V_3, V_4, V_5$. $R'$ and $R''$ are two distinct minimized MiCRs for $R$. $R'$ has two relational subgoals and $R''$ has three relational subgoals. Thus two distinct minimized MiCRs for a given CQAC MiCR can have a different number of relational subgoals. Note that our pruned-MiCR algorithm will output $R'$ (which is globally minimal) as opposed to $R''$ (which is a minimized MiCR but not a globally minimal minimized MiCR).

### 4.2 Decidability and Complexity

We now examine the complexity of the problem of computing containing rewritings. We show that for a class of CQAC problem inputs for which the homomorphism property is guaranteed to hold, it is NP-complete to determine whether a CQAC containing rewriting exists (Section 4.2). Further, we describe algorithm full+prined-MiCR and prove its correctness (Section 4.3).

We define the language $\mathbf{C} = \cup_{i=1}^{4} \mathbf{C}_i$, as a union of four sub-languages. $\mathbf{C}$ is a subclass of CQACs and shows good properties with respect to checking query containment. We denote by cLSI (oLSI, respectively) the closed (open, respectively) left-semi-interval ACs, and bycRSI (oRSI, respectively) the closed (open, respectively) right-semi-interval ACs.

**Definition 14. (Language C)** *We say that an AC is of SI type 1,2,3, or 4 if it is an cLSI, oLSI, cRSI, or oRSI AC, respectively. A query $Q$ is said to belong to the class $\mathbf{C}_i$, $i = 1, 2, 3, 4$ iff $Q$ is a CQ that does not have any ACs other than ACs of SI type $i$. We define $\mathbf{C}$=$\cup_{i=1}^{4}$ $\mathbf{C}_i$ and say that a* query is in class (or in language) $\mathbf{C}$ and is of type *$i$ if it belongs to class $\mathbf{C}_i$.*

Klug [22] has shown that when two queries are expressed in language $\mathbf{C}$ and both are of the same type then the homomorphism property holds between them [22].

We now present a theorem, which says that when queries and views are in one of the four classes: $\mathbf{C}_i$'s (defined in Section 2), then there exists a containing rewriting that is also in one of the four $\mathbf{C}_i$'s.

**Theorem 5.** *For a query $Q$ and a set of views $\mathcal{V}$, all of which belong to the class $\mathbf{C}_i$, the following holds: If there exists a containing rewriting $R$ of $Q$ using $\mathcal{V}$ such that $R$ is a UCQAC, then there also exists a containing rewriting $R'$ of $Q$ using $\mathcal{V}$, such that $R'$ belongs to the class $\mathbf{C}_i$.*

The proof is by construction of $R'$ from $R$.

*Proof.* Without loss of generality, we assume that the query $Q$ has only left-semi-interval ACs of the form $X < c$ or $X \leq c$. Construct a canonical database $D$ by freezing each variable in $Q$ to a unique constant and by populating $D$ with the resulting tuples of the relational subgoals of $Q$. Specifically, the variables that appear in the arithmetic predicates of $Q$ are frozen as follows. For any AC $(X\ \theta\ c)$ of $Q$ where $\theta$ is one of $<$ or $\leq$, freeze $X$ to a value $c - \epsilon_x$, where $\epsilon_x$ does not appear in $Q$ or in any view in $V$ and can be chosen to be arbitrarily small. Because the disjunctive rewriting $R$ contains $Q$, there must be a conjunct $R_1$ in $R$ that produces any tuple also produced by $Q(D)$ on the view instance $V(D)$, such that $R_1(D)$ is a superset of $Q(D)$. Therefore, there must be a containment mapping $\mu$ from the relational subgoals of $R_1$ to the predicates and constants in $D$ that produces the (frozen) head of $R_1$ on $D$. Because the mapping $\nu$ from the predicates and variables of $Q$ to the tuples and frozen constants in $D$ is bijective, a composition of $\mu$ with $\nu^{-1}$ is also a containment mapping from $R_1$ to the relational subgoals of $Q$.

We now construct from $R_1$ a new purely conjunctive rewriting $R_2$ by dropping any ACs in $R_1$. By construction, $R_1 \sqsubseteq R_2$. The query $Q$ and the set of views $V$ are in the same class $C_i$. All ACs in $R_2^{exp}$ come from the expansion of the views in $R_2$ and are in the same class $C_i$. Thus, we conclude that the homomorphism property holds between $R_2^{exp}$ and $Q$. (This conclusion follows from Theorem 4 in [42].)

Thus, to show containment of $Q$ in $R_2$, it remains to prove that $AC(Q) \Rightarrow AC(R_2)$. Recall that all ACs in $R_2$ are left-semi interval inequality comparisons of the form $X\ \theta\ k$ because they come from the views. We constructed the database $D$ by choosing the maximum values for the variables (that is, we chose each $c - \epsilon_x$ to be as large a value as possible). Computing $R_2$ on $D$ produces the (frozen) head of $R_2$. It follows that for each variable $X$ in $R_2$, such that $X$ maps to a variable $Y$ that occurs in an AC: $Y\ \theta\ c$ of $Q$, the range $(k)$ of $X$ is equal to or exceeds the maximum value $c$ for $Y$ in $D$, for otherwise computing $R_2$ on $D$ would not produce the (frozen) head of $R_2$. We conclude that $AC(Q) \Rightarrow AC(R_2)$. Thus, $R_2$ is $R'$ in the statement of the theorem.

(The only exception to $AC(Q) \Rightarrow AC(R_2)$ could be when $Q$ has an AC $Z \leq c$. Because $Z$ is frozen to $c - \delta$, it could be accepted by $R_2$ having an AC $T < c$, where $\mu(T) = Z$. In this case, the implication $(Z\ \leq\ c) \Rightarrow (Z\ <\ c)$ does not hold. However, this case is precluded by the condition that the ACs in $Q$ and $V$ belong to a single class $C_i$ and can thus only contain either $\leq$ or $<$ but not both. Therefore, $R_2$ must be having an AC $T \leq c$, where $\mu(T) = Z$. Now, the implication $(Z\ \leq\ c) \rightarrow (Z\ \leq\ c)$ holds trivially.)

**Theorem 6.** *Given a CQAC query and a set of CQAC views, it is decidable whether there exists a UCQAC containing rewriting that is a UCQAC. Furthermore, there exists an algorithm for computing a MiCR.*

We now turn our attention to special cases. We show that the problem of finding a containing rewriting of a query using views that are expressed in one of the four languages $\mathbf{C}_i$ indicated above is NP complete.

**Theorem 7.** *Given (1) a query $Q$ whose relational predicates belong to a set of predicates $P$, (2) a set of views $\mathcal{V}$ such that all views in $\mathcal{V}$ have relational predicates only from $P$, and given that (3) $Q$ and $\mathcal{V}$ are expressed using the same* *language $\mathbf{C}_i$ in $\mathbf{C}$, it is NP complete to decide whether there exists a containing rewriting of $Q$ using $\mathcal{V}$ in the language UCQAC.*

*Proof.* The proof that the problem is in NP uses the result of Lemma 2. If the answer to the question of whether there exists a containing rewriting is "yes", then the certificate is a containing rewriting together with a containment mapping from the expansion of the rewriting to the query. If the size of the rewriting is polynomial then the size of the containment mapping is also polynomial and checking that it is a containment mapping can be done in polynomial time because the homomorphism property holds. It remains to be proven that the size of the rewriting is polynomial. We do so in Lemma 2; see the statement and proof of the Lemma after the end of this proof.

NP-hardness is by reduction from CQ containment and is a consequence of the following theorem.

**Theorem 8.** *Let $Q$ be a CQ query, and let $\mathcal{V}$ be a set of CQ views. It is NP hard to decide whether there exists a safe containing rewriting in the language of UCQACs of $Q$ using $\mathcal{V}$.*

*Proof.* The problem is NP-hard: By reduction from CQ containment. Given queries $Q_1$ and $Q_2$, we construct $Q'_1$ and $Q'_2$: The head in both is $p(X)$ such that $p$ and $X$ do not appear in $Q_1$ and $Q_2$. The body of $Q'_1$ and $Q'_2$ each contain all the subgoals of $Q_1$ and $Q_2$ respectively and a new predicate $p'(X, t_i)$ where $t_1$ is the tuple of variables in the head of $Q_1$ and $t_2$ is the tuple of variables in the head of $Q_2$ respectively. Then $Q_2$ is contained in $Q_1$ iff $Q'_2$ has a containing rewriting which uses as view $Q'_1$. To prove, we need to argue that such a rewriting would use only one copy of the view and one copy of the query. This is easily shown by observing that $Q'_1$ is the only view. $Q'_1$ has the variable $X$ in the head. Thus, all subgoals in a containing rewriting must be $Q'_1(X)$ modulo variable renamings of $X$. Any rewriting must have $Q'_1(X)$ in the body in order to be safe because there is no other view and no different head homomorphisms of $Q'_1$ is possible. If the view has conjunctions of $Q'_1$ with itself, it can be rewritten with only one copy of the view. Similarly, if the query has multiple copies of $Q'_2$, it is equivalent to a query with one copy of $Q'_2$. This is again because $X$ is the only variable in the head of $Q'_2$, and thus, there is only one subgoal $Q'_2(X)$ that can be in the query, modulo variable renamings. The additional copies of $Q'_2(X)$ can be removed.

For safe containing rewritings, the following lemma says that it is possible to have a containing rewriting with a specific bound on its size:

**Lemma 2.** *If there exists a safe CQAC rewriting $R$ of a CQAC query $Q$, $Q \sqsubseteq R^{exp}$, then there exists a safe CQAC rewriting $R'$ of $Q$, $Q \sqsubseteq R'^{exp}$, such that the number of viewheads in $R'$ is less than or equal to the arity of the head of $Q$.*

*Proof.* We prove this by constructing the required $R'$. $R'$ is obtained from $R$ as follows: (1) drop all arithmetic predicates from R, and (2) for each variable $X$ that occurs in the head of $R$, choose to retain exactly one viewhead, say $V$, that occurs in the body of $R$, where viewhead $V$ is such that $X$ occurs in it. Drop all other viewheads from the body of $R$ to obtain the new $R'$. By construction, $R'$ has at most $n$ viewheads in its body, where $n$ is the number of unique distinguished variables that occur in $R$, that is, $n$ is basically the arity of the head of $R$, which is identical to the arity of the head of $Q$. Furthermore, we know that $R'$ is safe, since the process of construction ensures that each variable that occurs in the head of $R'$ also occurs in one of the viewheads in the body of $R'$. Now, the subgoals in the body of $R'$ are a subset of the subgoals in the body of $R$, hence the subgoals in the body of $R'^{\,exp}$ are also a subset of the subgoals in the body of $R^{exp}$. There is a identity containment mapping $I$ from the body of $R'^{exp}$ to that of $R^{exp}$. Because, $Q \sqsubseteq R^{exp}$, there exist containment mappings $\mu_i$ from $R^{exp}$ to $Q$, such that

$$AC(Q) \Rightarrow \cup_i \mu_i(AC(R^{exp})). \tag{1}$$

By composing $I$ with $\mu_i$, we get containment mappings $\nu_i = \mu_i$ (because $I$ is an identity mapping) from $R'^{exp}$ to $Q$. Now, we need to show $AC(Q) \Rightarrow \cup_i(AC(R'^{exp}))$ to establish that $Q \sqsubseteq R'^{exp}$. Note that if an AC appears in $R'^{exp}$, it also appears in $R^{exp}$ (by construction). Therefore, from Equation 1, we have $Q \sqsubseteq R'^{exp}$ and thus, $Q \sqsubseteq R'^{exp}$.

For all boolean queries, the query $Q() : -$, which says that $Q$ is always true, is a safe containing CQAC. Note that this is consistent with Lemma 2 above, because the number of variables in the head of $Q$ and the number of subgoals in its body is the same (both are zero).

It turns out that a MiCR is unique up to equivalence as expansions.

**Theorem 9.** *Under the CWA, and for queries and views in the same language $C_i$ in $C$, the MiCR is unique up to equivalence as expansions.*

*Proof.* Let $R_1$ and $R_2$ be two UCQAC MiCRs of query $Q$ using the set of views $\mathcal{V}$, such that $R_1$ and $R_2$ are not equivalent as expansions. Since $R_1$ is *minimally* containing, by definition $R_2^{exp}$ is not contained in $R_1^{exp}$. Similarly, $R_1^{exp}$ is also not contained in $R_2^{exp}$. Consider all canonical databases of $Q$. For each such canonical database $D_i$, let $t_i$ be the corresponding tuple obtained from the head of $Q$. Now let $r1_i$ be a CQAC in the UCQAC $R_1$, such that $r1_i$ produces $t_i$ on $D_i$. Note that since $R_1$ is a *containing* rewriting, such $r1_i$ must exist. Similarly let CQAC $r2_i$ in UCQAC $R_2$ produce $t_i$ on $D_i$. Since the query and views are in a language in $C$, there must be a single mapping from the variables and constants in $r1_i^{exp}$ to the constants in $D_i$ that produced $t_i$. Moreover since $D_i$ is obtained by freezing the variables in $Q$ to corresponding constants, there must also exist a corresponding mapping from the variables and constants in $r1_i^{exp}$ to the variables and constants in $Q$. Let $\mu 1_i$ be this mapping from $r1_i^{exp}$ to $Q$. Similarly, let $\mu 2_i$ be the mapping from $r2_i^{exp}$ to $Q$. Replace each distinguished variable in $r1_i$ by its image under $\mu 1_i$ and each distinguished variable in $r2_i$ by its image under $\mu 2_i$ and construct the CQAC $r_i$ by conjoining the bodies of $r1_i$ and $r2_i$. $D_i$ specifies a total order on the variables and constants in $Q$. Depending upon which of these variables and constants appear in $r_i$, add to $r_i$

all applicable ACs that are derived from $D_i$ and that involve the variables and constants in $r_i$. If $D_i$ specifies that two variables in the head of $r_i$ are equal, then equate those variables. The $r_i$ thus obtained is a containing rewriting of $Q$ that is contained in each of $r1_i$ and $r2_i$. Let the UCQAC $R$ be obtained by taking the union of the $r_i$'s constructed for all possible $D_i$'s. Thus $R$ is a containing rewriting of $Q$ using $\mathcal{V}$, but it is contained in both $R_1$ and $R_2$, which contradicts our assumption that rewritings $R_1$ and $R_2$ are *minimally* containing.

Theorem 9 shows that for a given query and set of views, if $R_1$ and $R_2$ are both MiCRs, then it is necessary that $R_1$ and $R_2$ are equivalent as expansions. However, note that for contained rewritings, if $R_1$ and $R_2$ are both MCRs, then it is not necessary that $R_1 \equiv R_2$.

## 4.3 The full+pruned-MiCR Algorithm

In this subsection, we discuss in detail the working of the full-MiCR, pruned-MiCR, and CB-MiCR algorithms. The bucket-construction procedure used by pruned-MiCR is further described through the pseudocode in Algorithm 2 and through Example 11.

### The pruned-MiCR Algorithm

The pruned-MiCR algorithm accepts on input a CQAC query $Q$, and a set of CQAC views $\mathcal{V}$, and returns CQACs $R$ and $R'$ defined in terms of $\mathcal{V}$, such that $R$ is a MiCR and $R'$ is a minimized MiCR of $Q$ using $\mathcal{V}$. Since pruned-MiCR uses a bucket strategy that constructs only some of the buckets (rather than constructing all buckets) it may not always be able to find the minimized version $R'$ of the MiCR. However, it always finds the MiCR $R$ itself, on all inputs. Algorithm 2 gives the pseudocode of pruned-MiCR and Example 11 illustrates the bucket forming strategy that it uses.

The pruned-MiCR algorithm starts by initializing $R$, so that its head is identical to the head of $Q$, and its body contains no subgoals. Then it examines one-by-one each view $V$ in $\mathcal{V}$. For each containment mapping $\mu$ from the relational subgoals in $V$ to the relational subgoals in $Q$, pruned-MiCR constructs $h(V)$ from $V$, where the function $h$ is such that it replaces each distinguished variable $X$ in $V$, by its image $\mu(X)$ under mapping $\mu$. The algorithm then constructs $ac$, which is a conjunction of some ACs from $Q$. An AC from $Q$ is included in $ac$ iff all variables in that AC appear in $h(V)$. It then checks the implication $AC(Q) \Rightarrow \mu(AC(h(V)))$, that is, it checks if the ACs of $Q$ imply the ACs of $h(V)$. If this is true, it conjoins all the subgoals in $h(V)$ and the subgoals in $ac$, with the subgoals that are already in $R$, thus forming a new $R$. Once all the given views have been processed in this way, the algorithm checks if every head variable of $Q$ is available in at least one relational subgoal in the body of $R$. If this is true, then the CQAC whose body consists of the existing subgoals in $R$, and whose head is identical to the head of $Q$, is a MiCR. We call this the *full*

*MiCR* of $Q$ using $\mathcal{V}$, and the part of the algorithm till this point is also called the *full MiCR algorithm*. Otherwise, the algorithm declares that there exists no safe containing rewriting of $Q$ using $\mathcal{V}$, and stops.

If a full MiCR $R$ has been found, the pruned-MiCR algorithm proceeds with its bucket-forming strategy. For every relational subgoal $g_r$ in every $h(V)$ in $R$, and for every subgoal $g_q$ in $Q$ that $g_r$ maps to, pruned-MiCR tries to insert $V'$, which is the head of that $h(V)$ conjoined with the $ac$ for that $h(V)$, into appropriate buckets as follows. Pruned-MiCR compares $g_r$ with $g_{r_i}$, for all $g_{r_i}$'s for which it has previously constructed a bucket ($g_q$, $g_{r_i}$) involving the same query subgoal $g_q$. If any $g_{r_i}$ is strictly more restrictive than $g_r$ (i.e. if $g_r$ can map to $g_{r_i}$ in a containment mapping), then the flag $constructNewBucket$ for subgoal $g_r$ (which was initialized to true), is now made false. However, if instead for any $g_{r_i}$, it finds that $g_r$ is strictly more restrictive than that $g_{r_i}$, then the bucket ($g_q$, $g_{r_i}$) for that $g_{r_i}$ is deleted. Finally, if there exists some $g_{r_i}$ which is exactly as restrictive as $g_r$, then $V'$ is inserted into the existing bucket ($g_q$, $g_{r_i}$) and flag $constructNewBucket$ is made false[16]. In the end, if $constructNewBucket$ is still true, then pruned-MiCR creates a new bucket denoted by ($g_q$, $g_r$), and inserts $V'$ into that new bucket. Once pruned-MiCR has finished processing all relational subgoals in all $h(V)$'s, it runs the minimum-set-cover algorithm over all the buckets. Let $R'$ be the conjunction of exactly those $V'$s that are in the smallest set of $V'$s that can represent all buckets (i.e. $R'$ is a conjunction of the $V'$s that are returned as answer by the minimum-set-cover algorithm). If the expansion of $R'$ is contained in the expansion of the full MiCR $R$, then the pruned-MiCR algorithm is able to provide the user with $R'$, which is the minimized MiCR of $Q$ using $\mathcal{V}$. Otherwise, it provides the user with only the full MiCR $R$ that was obtained earlier.

### The CB-MiCR Algorithm

The pruned-MiCR algorithm is able to find the full MiCR on all inputs and is also able to find the minimized MiCR on some inputs. In order to have an algorithm which also finds the minimized MiCR on all inputs, we developed the CB-MiCR algorithm, which does a simple exhaustive search of all subsets of the set of relational subgoals in the full MiCR. The obvious tradeoff involved is that CB-MiCR, on account of the exhaustive search that it does to guarantee the generation of a minimized MiCR, typically runs slower than pruned-MiCR, which uses a more sophisticated bucket-strategy.

Both the CB-MiCR algorithm and the pruned-MiCR algorithm, initially apply the same steps (i.e. the full MiCR algorithm) that are described in Section 4.3 in the context of pruned-MiCR. However, once the full MiCR $R$ has been found, the two algorithms apply different strategies in obtaining the minimized MiCR

[16] However, if the head of the $h(V)$ which corresponds to $g_r$ contains at least one head variable which is not present in the head of the $h(V)$ which corresponds to $g_{r_i}$, then $constructNewBucket$ is not made false, and $V'$ is not inserted into that bucket.

$R'$ from $R$. CB-MiCR adopts the naive approach of forming candidates by looking at all possible subsets of the relational subgoals in $R$. It considers these subsets in increasing order of the size of the subsets. If any candidate has an expansion that is equivalent to the expansion of the full MiCR, then the CB-MiCR algorithm declares this candidate to be the minimized MiCR $R'$, and stops. In the worst case, CB-MiCR has to consider all subsets up to the largest subset, which consists of all the subgoals from $R$. This is the case in which the minimized MiCR happens to be the same as the full MiCR, that is, $R'$ is $R$ itself.

**An example illustrating bucket construction in pruned-MiCR**

*Example 11.*

$Q(X, Z)$ :- $p(X, Y, Y, X, X),\ s(Z, Z)\ Z < 3$.
$V_1()$ :- $p(X_1, A_1, B_1, X_1, C_1)$.
$V_2(X_2)$ :- $p(X_2, A_2, B_2, X_2, C_2)$.
$V_3(X_3)$ :- $p(X_3, A_3, B_3, X_3, X_3)$.
$V_4(X_4)$ :- $p(C_4, A_4, B_4, X_4, X_4)$.
$V_5(B_5)$ :- $p(A_5, Y_5, Y_5, B_5, C_5)$.
$V_6(Z_6)$ :- $s(Z_6, T_6)$.
$R(X, Z)$ :- $V_1(),\ V_2(X),\ V_3(X),\ V_4(X),\ V_5(X),\ V_6(Z),\ Z < 3$.
$R'(X, Z)$ :- $V_3(X),\ V_5(X),\ V_6(Z),\ Z < 3$.

In Example 11, suppose we are trying to find the full MiCR and the minimized MiCR of the query $Q$ using the set of views $\mathcal{V} = \{V_1, V_2, V_3, V_4, V_5, V_6\}$.

*Computing the full MiCR*

As outlined in Section 4.3 and in Algorithm 2, the pruned-MiCR algorithm initializes $R$ so that its head is identical to the head of $Q$ and its body contains no subgoals (that is, $R$ is initially set to $R(X, Z)$ :–). Pruned-MiCR then examines one-by-one all views in $\mathcal{V}$. For each view $V$ in $\mathcal{V}$, it tries to discover all possible $h(V)$'s resulting from this $V$ that can be added to $R$. In Example 11, on examining $V_1$, $V_1()$ gets added to $R$. Similarly, for $V_2$ the pruned-MiCR algorithm adds $V_2(X)$. Also, for $V_3$, $V_4$, and $V_5$ it adds $V_3(X)$, $V_4(X)$, and $V_5(X)$, respectively. For $V_6$ it adds $V_6(Z)$ along with the AC $Z < 3$. Thus, the resulting $R$ has six relational subgoals and one arithmetic subgoal. Note that $Q$ has two variables in the head, $X$ and $Z$, and each of these is available in at least one relational subgoal in the body of $R$. Thus, $R$ is the full MiCR of $Q$ using $\mathcal{V}$.

*Forming the buckets*

$V_1$: For view $V_1$, for the view subgoal $p(X_1, A_1, B_1, X_1, C_1)$ and the query subgoal $p(X, Y, Y, X, X)$, pruned-MiCR constructs a new bucket and inserts $V_1()$ as an entry in that bucket.

$V_2$: For view $V_2$, for the view subgoal $p(X_2, A_2, B_2, X_2, C_2)$ and the query subgoal $p(X, Y, Y, X, X)$, pruned-MiCR constructs a new bucket and inserts $V_2(X)$

**Algorithm 2:** The full+pruned-MiCR algorithm.

```
Input  : CQAC query $Q$; set of CQAC views $\mathcal{V}$
Output : the full MiCR $R$ of $Q$ in terms of $\mathcal{V}$, the globally minimal minimized MiCR $R'$
1. //Construct the full MiCR $R$
2. Initialize $R$ to have a head that is identical to the head of $Q$ and a body that is empty;
3. for each view $V$ in $\mathcal{V}$ do
    4. for each containment mapping $\mu$ from the relational subgoals in $V$ to
    the relational subgoals in $Q$ do
        5. construct $h(V)$ by replacing each distinguished variable $X$ in $V$ by $\mu(X)$;
        6. $ac \leftarrow null$;
        7. for each AC $ac_i \in AC(Q)$ do
            8. if all variables in $ac_i$ appear in $h(V)$ then
                9. $ac \leftarrow ac \wedge ac_i$;
        10. if $AC(Q) \Rightarrow \mu(AC(h(V)))$ then
            11. conjoin $h(V), ac$ with the existing body of $R$;
12. if there is some variable in the head of $Q$ that is not present in any
relational subgoal in $R$ then
    13. output that there exists no safe MiCR of $Q$ using $\mathcal{V}$ and stop;
14. //Construct buckets for approximate containment checking
15. for each relational subgoal $g_r$ in each $h(V)$ in $R$ do
    16. for each subgoal $g_q$ in $Q$ that $g_r$ maps to do
        17. $V' \leftarrow h(V), ac$;  $constructNewBucket \leftarrow true$;
        18. for every bucket $(g_q, g_{r_i})$ that already exists for $g_q$ do
            19. if $g_{r_i}$ is more restrictive than $g_r$ then
                20. $constructNewBucket \leftarrow false$;
            21. if $g_r$ is more restrictive than $g_{r_i}$ then
                22. delete the bucket $(g_q, g_{r_i})$;
            23. if $g_{r_i}$ and $g_r$ are equally restrictive then
                24. insert $V'$ in the existing bucket $(g_q, g_{r_i})$;
                25. $constructNewBucket \leftarrow false$;
        26. if $constructNewBucket$ is true then
            27. create a new bucket $(g_q, g_r)$ with one entry $V'$ in it;
28. Run a minimum-set-cover algorithm over all the buckets to select a smallest-sized
set of $V'$ entries such that at least one entry from each bucket has been selected;
29. Construct CQAC $R'$ by taking a conjunction of all the selected $V'$ entries;
30. if $(R')^{exp} \sqsubseteq R^{exp}$ then
    31. Output $R$ as the full MiCR and $R'$ as the globally minimal minimized MiCR;
32. else
    33. Output $R$ as the full MiCR;
```

as an entry in that bucket. Although $V_2$ covers the $p$-subgoal in $Q$ exactly as tightly as $V_1$ covers it, that is, although both $V_1$ and $V_2$ are equally restrictive, $V_2$ contributes the additional head variable $X$ which is not contributed by $V_1$. Hence, the pruned-MiCR algorithm constructs a new bucket, rather than inserting $V_2(X)$ in the same bucket that already contains $V_1()$.

$V_3$: For view $V_3$, for the view subgoal $p(X_3, A_3, B_3, X_3, X_3)$ and the query subgoal $p(X, Y, Y, X, X)$, pruned-MiCR constructs a new bucket and inserts $V_3(X)$ as an entry in that bucket. It also deletes the previous two buckets that it has created, since $V_3$, being more restrictive than $V_1$ and $V_2$, covers the $p$-subgoal in $Q$ more tightly than either of them.

$V_4$: For view $V_4$, for the view subgoal $p(C_4, A_4, B_4, X_4, X_4)$ and the query subgoal $p(X, Y, Y, X, X)$, pruned-MiCR does not construct any new bucket since the existing bucket which contains $V_3(X)$ already covers the $p$-subgoal in $Q$ in a tighter way.

$V_5$: For view $V_5$, for the view subgoal $p(A_5, Y_5, Y_5, B_5, C_5)$ and the query subgoal $p(X, Y, Y, X, X)$, pruned-MiCR constructs a new bucket and inserts $V_5(X)$ as an entry in that bucket. The MiCR algorithm sees that currently there is only one bucket for the $p$-subgoal in $Q$. But $V_5(X)$ has the same value for the second and third arguments in its $p$-subgoal, and so it covers a constraint which the existing bucket does not cover. Hence a new bucket is created and $V_5(X)$ is inserted into it.

$V_6$: For view $V_6$, for the view subgoal $s(Z_6, T_6)$ and the query subgoal $s(Z, Z)$, pruned-MiCR constructs a new bucket and inserts $V_6(Z),\ Z < 3$ as an entry in that bucket.

*Minimal Set Cover and Finding the Minimized MiCR*

At the end of the bucket-forming procedure, finally there are three buckets and the minimum-set-cover routine returns $\{V_3(X),\ V_5(X),\ V_6(Z),\ Z < 3\}$ as answer, so pruned-MiCR constructs

$R'(X, Z)$ :- $V_3(X),\ V_5(X),\ V_6(Z),\ Z < 3$.

Since $R'$ is obtained by deleting some subgoals from $R$, we know that $R \sqsubseteq R'$, and hence $R^{exp} \sqsubseteq (R')^{exp}$. On checking for containment in the other direction, pruned-MiCR finds that $(R')^{exp} \sqsubseteq R^{exp}$ is also true. Thus, $(R')^{exp} \equiv R^{exp}$, and so pruned-MiCR returns $R'$ as the globally minimal minimized MiCR.

The CB-MiCR algorithm works identical to the pruned-MiCR algorithm except that instead of constructing and using buckets, it considers all subsets of the relational subgoals of $R$. It first considers all subsets of size one, and does not find any rewriting whose expansion is equivalent to the expansion of $R$. It then considers all subsets of size two, and so on. When it is considering subsets of size three, it considers $\{V_3(X),\ V_5(X),\ V_6(Z),\ Z < 3\}$, and determines that the expansion of this rewriting is equivalent to the expansion of the full MiCR. Hence, it returns

$R'(X,Z)$ :- $V_3(X),\ V_5(X),\ V_6(Z),\ Z < 3$

as the minimized MiCR.

We now show an example where our algorithm is incomplete. For example, consider the query $Q$ and the four views $V1, V2, V3$, and $V4$.

*Example 12.*

$Q$ :- $p(X,X),\ r(X,X)$.

$V1(X)$ :- $p(X,X)$.

$V2(X)$ :- $p(X,Y),\ r(X,Y)$.

$V3()$ :- $r(X,X)$.

$V4()$ :- $p(X,Y)$.

$R$ :- $V1(X),\ V2(X),\ V3(),\ V4()$.

$R$ is a MiCR of $Q$. Notice that $R$ is not a minimized MiCR because $V4$ is redundant in $R$. The equality of the variables in the $p$ subgoal is covered by $V1$, the equality of the variables in the $r$ subgoal is covered by $V3$, and the equality of the first variable in the $p$ subgoal with the first variable in the $r$ subgoal and the equality of the second variable in the $p$ subgoal and the second variable in the $r$ subgoal are covered by $V2$. However, because the pruned-MiCR algorithm only checks for shared variable constraints within a subgoal (due to the process of bucket construction), it will deem $V1$ to be covering the p-subgoal more tightly than $V2$ and $V3$ to be covering the r-subgoal more tightly than $V2$ and $V2$ will not be entered in either bucket. Similarly, view $V1$ covers the p-subgoal more tightly than view $V4$ and thus $V4$ is deleted. Consequently, the pruned-MiCR algorithm generates $R' : -V1(X), V3()$ as a minimization of $R$ and checks for the equivalence of $R$ and $R'$. Because they are not equivalent, the algorithm will undo our attempted minimization, reject $R'$ and return the non-minimal $R$.

## Complexity of pruned-MiCR

We now discuss the complexity of the pruned-MiCR algorithm. Let $r$ be the maximum number of relational subgoals in $Q$ and in any view in $\mathcal{V}$. Let $p$ be the number of relational subgoal in the full-MiCR $R$ that is given as input to pruned-MiCR. Thus the expansion of $R$ has at most $p \times r$ subgoals. Hence the maximum possible number of buckets is $p \times r \times r$. The simplest minimum-set-cover routine which examines all subsets of the set of buckets takes at most $2^{p \times r \times r}$ time. The algorithm needs to perform a total of just *one* containment test (as its final step), so this time can be taken as constant. Thus, the overall time is $O(2^{p \cdot r \cdot r})$.

## 4.4 Correctness of the MiCR algorithm

**Theorem 10.** *The* full-MiCR *part of algorithm* full+pruned-MiCR *is sound and complete for all problem inputs for which the homomorphism property holds between the expansion of the rewriting and the input query.*

Note that the full MiCR algorithm is applicable only in cases where the homomorphism property holds between the expansion of the rewriting and the query.

**Theorem 11. (Soundness Theorem for Full-MiCR)** *Given a CQAC query $Q$ and CQAC views $\mathcal{V}$. For any CQAC $R$ (defined in terms of $\mathcal{V}$) that is output by the full-MiCR subroutine, $R$ is a MiCR of $Q$ using $\mathcal{V}$.*

*Proof.* Let $R$ be a rewriting generated (after line 11) by the full-MiCR algorithm. We need to show that $R^{exp}$ contains $Q$. We show that there exists a containment mapping $\mu$ from $R^{exp}$ to $Q$ that satisfies $AC(Q) \Rightarrow \mu(AC(R^{exp}))$. The mapping $\mu$ can be constructed by composing the individual partial mappings in line 4. There is a containment mapping $\mu_i$ from each $h(V_i)$ added to $R$ constructed in line 4. By construction, each distinguished variable in $h(V_i)$ that appears in $R$ is replaced by $\mu_i(X)$. Thus, $\mu$ maps each variable in $R$ to itself in $\mu(R^{exp})$. Each non-distinguished variable $Y$ in each $h(V_i)$ is unique (does not appear in any other $h(V_i)$) and $\mu(Y) = \mu_i(Y)$. Thus, $\mu$ is a containment mapping from $R^{exp}$ to $Q$.

Now, we need to show $AC(Q) \Rightarrow \mu(AC(R^{exp}))$. The arithmetic predicates in $AC(R^{exp}$ either (i) appear in $R$ or (ii) are obtained by the expansion of a view in $R$. (i) By construction, each arithmetic comparison $ac$ appearing in $R$ is obtained from $AC(Q)$ (lines 7-9), and $\mu$ maps all variables appearing in $R$ to themselves. Thus, $AC(Q) \Rightarrow \mu(ac)$. (ii) Each AC, $a_j$ appearing in the expansion of $R$ but not in $R$ must have been obtained from the expansion of some view in $R$. For each $h(V_i)$ in $R$, in (line 10-11), the algorithm checks that $AC(Q) \Rightarrow \mu_i(AC(h(V_i))$ for all $i$. Again, for each $X$ appearing in the expansion of the rewriting $R$, $\mu(X) = \mu_i(X)$ by construction as shown in the previous paragraph. And, $\mu_i(a_j)$ appears in $\mu_i(h(V_i)$ (for some $i$) and is implied by $AC(Q)$. Thus, $AC(Q) \Rightarrow \mu(a_j)$. Therefore, all ACs in $R^{exp}$ are implied by $AC(Q)$.

**Theorem 12. (Completeness Theorem for Full-MiCR)** *Given a CQAC query $Q$ and CQAC views $\mathcal{V}$, let $P$ be a CQAC query defined in terms of $\mathcal{V}$ such that there exists a single containment mapping $\mu_P$ from $P^{exp}$ to $Q$ that establishes $Q \sqsubseteq P^{exp}$. Then (1) the output of the full-MiCR subroutine is not empty, and (2) denoting by $R$ the CQAC output of the full-MiCR subroutine we have that $R^{exp} \sqsubseteq P^{exp}$.*

*Proof.* 1. Let $s$ be a relational subgoal in $P$. Let $\mu_s$ be the mapping which is obtained from $\mu_P$ by restricting its domain to those subgoals in $P^{exp}$ which are in the expansion of $s$. Thus, $\mu_s$ is a containment mapping from the expansion of $s$ to $Q$. The predicate name of $s$ is the name of one of the views, say view $v$, in $\mathcal{V}$. Since the full-MiCR subroutine examines each view in $\mathcal{V}$ (line 3 in Algorithm

2) and each possible containment mapping (line 4 in Algorithm 2), it must have examined $v$ and must have added $s$ as a relational subgoal in $R$. This proves (1).

2. Now, let $ac_s$ be the ACs that are conjoined with $s$ while adding it to $R$ (line 11 in Algorithm 2). $ac_s$ are a subset of the ACs of $Q$. Hence, the expansion of $s$ conjoined with $ac_s$ forms a CQAC that contains $Q$. This reasoning is true for every relational subgoal $s$ in $P$, therefore $R^{exp} \sqsubseteq P_0^{exp}$, where $P_0$ is a conjunction of all the relational subgoals in $P$.

3. Since, $Q \sqsubseteq P^{exp}$, the ACs in $P$ are not more restrictive than the ACs of $Q$, that is for any variable, say $X$, in $P^{exp}$, the values which $X$ is allowed to take in $P^{exp}$ cannot be more restrictive than the values that the image of $X$ under $P$ is allowed to take in $Q$. Also, the extra ACs that are added to $R$ (line 11 in Algorithm 2) are taken exactly as they are from the body of $Q$. Hence, the values which $X$ is allowed to take in $P^{exp}$ cannot be more restrictive than the values of the corresponding variable in $R^{exp}$ as well. Thus, $R^{exp} \sqsubseteq P^{exp}$.

**Theorem 13. (Soundness Theorem for Pruned-MiCR)** *Given a CQAC query $Q$, CQAC views $\mathcal{V}$, and CQAC $R$ ($R$ is defined in terms of $\mathcal{V}$ and there exists a single containment mapping $\mu_R$ from $R^{exp}$ to $Q$ that establishes $Q \sqsubseteq R^{exp}$) that is output by the full-MiCR subroutine. For any CQAC $R'$ defined in terms of $\mathcal{V}$ that is output as the globally minimal minimized MiCR by the pruned-MiCR algorithm (1) $(R')^{exp} \equiv R^{exp}$, and (2) there does not exist any CQAC $R''$ defined in terms of $\mathcal{V}$ for which $(R'')^{exp} \equiv R^{exp}$ and for which the number of relational subgoals in $R''$ is strictly less than the number of relational subgoals in $R'$.*

*Proof.* 1. By construction, the head of $R'$ is identical to the head of $R$ and the subgoals in $R'$ are a subset of the subgoals in $R$. Hence, $R^{exp} \sqsubseteq (R')^{exp}$. Since $R'$ is output by the pruned-MiCR algorithm, it has been explicitly checked (in line 30 of Algorithm 2) that $(R')^{exp} \sqsubseteq R^{exp}$ is true. Hence, $(R')^{exp} \equiv R^{exp}$ and (1) is proved.

2. The proof of (2) follows from the correctness of the minimum-set-cover algorithm.
(i) At the end of the bucket-forming procedure of the pruned-MiCR algorithm, each bucket represents a condition[17] or requirement that holds true in the full MiCR, and must therefore also hold true in a minimized MiCR. Suppose that when picking entries from buckets in order to construct the minimized MiCR, we do not pick any entry from a particular bucket $B$. Then the condition represented by bucket $B$ will not hold in the resulting rewriting. Hence, the resulting rewriting will not be a MiCR since its expansion will not be equivalent to the expansion of the full MiCR.

[17] For example, for a bucket labeled $(p(X, X, Y), p(A, A, B))$ the condition is that the first two arguments of the $p$-subgoal have the same value.

(ii) The minimum-set-cover algorithm picks not just any set cover, but it picks the set cover of the *smallest* possible size. That is, the answer returned by the minimum-set-cover algorithm consists of the smallest possible number of bucket entries that are needed in order to cover all buckets.

Therefore, if we are looking for a rewriting, say $R''$, such that $R''$ that has at least one entry from each bucket, and such that the number of relational subgoals in the body of $R''$ is strictly less than the number of relational subgoals in the body of $R'$, then by (i) and (ii) above, we conclude that such an $R''$ does not exist.

Thus, when the pruned-MiCR outputs a minimized version which is different from the full MiCR, then by definition this is the *globally minimal minimized* CQAC MiCR for the given query $Q$ using the given viewset $\mathcal{V}$.

**Theorem 14. Completeness Theorem for pruned-MiCR (in the MiCR sense)** *Given a CQAC problem input* $(Q, \mathcal{V}, R)$*, where R is a CQAC MiCR for Q using* $\mathcal{V}$*, let* $R'$ *be the (CQAC) output of algorithm pruned-MiCR. Then* $R'$ *is a CQAC MiCR for Q and* $\mathcal{V}$*.*

While being complete in the sense of Theorem 14, algorithm pruned-MiCR is not complete in the sense that it does not always produce a (globally) *minimized* MiCR for its problem inputs. The reason for this behavior is that as discussed in Section 4.3 pruned-MiCR does not consider shared variables across query subgoals (that is, variables that occur in two or more subgoals of the query) while minimizing the MiCR.

## 5 Variable Types

Checking for containment of two conjunctive queries with arithmetic predicates requires constructing all possible total orders of the variables and constants in the two queries' canonical databases. For $n$ variables and constants, the number of total orders is $O(n!)$. In practice, even for a very reasonably sized query having 10 variables and constants, the number of total orders becomes prohibitively large. In this subsection, we explore the question whether we can reduce the number of total orders that must be checked while checking for containment among CQACs. The reduction is based on the intuition that it may not make sense to compare a variable representing "price" and another representing, say, "date", e.g., "Price1 > Date1". We explain our observations below after introducing some required terminology.

We obtain the *type* of each variable as explained below. For each argument of each predicate appearing in the body of either query, we assign a distinct equivalence class. Merge the equivalence classes of arguments that have a shared variable. If variable $X$ and $Y$ appear in the same AC, then merge the two equivalence classes of the arguments where $X$ and $Y$ appear. We refer to the final merged equivalence classes as the *types* of each argument and its corresponding variable.

Let $Q1$ and $Q2$ be two CQACs. Recall that $Q1$ is contained in $Q2$ iff: $\beta_2 \Rightarrow \mu_1(\beta_1) \vee \mu_2(\beta_1) \dots$ The *projection* of this implication on a type $t$ is obtained by computing the closure of each of the conjuncts in the lhs and rhs of the implication and eliminating all ACs that contain variables not of type $t$ from the implication. We say an AC is of type $i$ only if all variables appearing in the AC is of type $i$.

**Lemma 3.** *In the implication $I : \beta_2 \Rightarrow \mu_1(\beta_1) \vee \mu_2(\beta_1) \dots$, if the rhs of $I$ contains an AC of type $i$, then each disjunct in the rhs of $I$ contains an AC of type $i$.*

*Proof.* Each disjunct on the rhs of $i$ is a mapping of the ACs in a single query. If the rhs of $I$ contains an AC of type $i$, that must have come from one of the disjuncts. If one of the disjuncts has an AC of type $i$, then the query body has an AC of type $i$ and the mapping of this AC under the different mappings must produce an AC of type $i$ in each disjunct on the rhs.

**Theorem 15.** *If $I : \beta_2 \Rightarrow \mu_1(\beta_1) \vee \mu_2(\beta_1) \dots$ holds, then $\forall i, projection(I, i)$ holds where $i$ is a merged equivalence class constructed from $Q_1$ and $Q_2$.*

*Proof.* Consider a particular type $i$. On the lhs of $projection(I, i)$, we have all ACs in the closure of $\beta_2$ that are of type $i$. On the rhs of the projection, we have all ACs of type $i$ in the closure of the rhs conjuncts in I. Assume that $projection(I, i)$ does not hold. Then, there exists one canonical database on which the lhs of $projection(I, i)$ holds but none of the disjuncts in its rhs holds. Along with Lemma 3, this case would imply that there is an AC of type $i$ in each of the disjuncts of the rhs that does not hold and each disjunct in the rhs of $I$ has an AC of type $i$ that does not hold. The last statement implies that $I$ does not hold — a contradiction.

Theorem 15 says that when $I$ holds, its projection implications on each type holds, however, the reverse is not true. The intuition to why the reverse does not hold is illustrated by the following example.

*Example 13.* Consider the two queries:
$Q1 : -p(X, Y), X < 25, Y < 0.$
$Q2 : -p(X1, X2), p(X3, X4), X1 < 25, X2 = 0, X3 = 25, X4 < 0.$

The implication $I$:

$$(X1 < 25, X2 = 0, X3 = 25, X4 < 0) \Rightarrow ((X1 < 25, X2 < 0) \vee (X3 < 25, X4 < 0))$$

clearly does not hold because $X2 = 0$ violates the first disjunct on the rhs and $X3 = 25$ violates the second.

Now, consider the query $Q1$. The two types in it are type($X$) and type($Y$). $X$, $X1$, $X3$ are of type($X$) and $Y$, $X2$, $X4$ are of type($Y$). Projecting $I$ on type($X$) produces:

$$(X1 < 25, X3 = 25) \Rightarrow (X1 < 25) \vee (X3 < 25)$$

which holds. Similarly, projecting $I$ on type($Y$) produces

$$(X2 = 0, X4 < 0) \Rightarrow (X2 < 0) \vee (X4 < 0)$$

which holds too.

As this example illustrates, even if all the projection implications on all the types hold, it does not necessarily imply that the full implication holds.

In case the workload contains many queries where the containment check fails, the corollary of Theorem 15, Corollary 2 stated below can be used to resolve these cases faster. Instead of performing the implication check on the full implication, we can check whether the implication holds on each type. If the implication does not hold on any one type, then the containment of the two queries does not hold.

**Corollary 2.** *If projection(I,i) does not hold for any type $i$, where $I$ is the implication that must be checked to prove the containment of $Q2$ in $Q1$, $Q2$ is not contained in $Q1$.*

*Proof.* The corollary follows from Theorem 15. If $I$ was true, then $projection(I, i)$ for all types $i$ would be true. Because we know that one $projection(I, i_0)$, say, is not true, then $I$ must not be true. If $I$ is not true, the containment does not hold.

Example 13 shows that even when the rhs of the projection of the implication on each type holds, the implication itself does not hold. However, if $I$ has only one AC in the rhs, i.e., if the homomorphism property holds between the two queries, then the theorem would hold because all the ACs that hold in the rhs of the projected implications belong together in one disjunct (because the rhs contains only one disjunct when the homomorphism property holds). Thus, in this special case, we can check whether the projection of the implication holds for all types and decide containment based on that.

**Theorem 16.** *The implication $I : \beta_2 \Rightarrow \mu(\beta_1)$ holds iff $\forall i, projection(I, i)$ holds for each type $i$ in the queries whose containment is being checked.*

*Proof.* Theorem 15 already proves that if $I$ holds then all the projections of $I$ for each type $i$ in the queries hold. Let all the projections of $I$ for each type $i$ hold. For any canonical database where the lhs of $I$ holds, the lhs of each projection of $I$ holds too. Thus, the respective rhs of each projection of $I$ holds. Take the conjunction of all the rhs of these individual projections. This conjunction is nothing but the rhs of $I$ and because each conjunct holds, their conjunction holds. Thus the rhs of $I$ also holds and therefore, $I$ holds.

Thus, in our work (see, e.g., the full+pruned MiCR algorithm in Section 4.3), where we assume the homomorphism property holds, we can project the implication on each type and check if the projected implications hold and based on that decide on the containment of the queries. Because of this improvement, our algorithm is significantly more scalable than that using a normal containment checking algorithm.

The techniques identified in this section can be used to reduce the time by any algorithm that checks for containment among two CQACs. For example, in our MiCR algorithm, the technique can be applied while containment checks arise in the following steps: (1) While constructing the full MiCR, the view body is being checked for containment in the query body, and (2) Before generating the minimized candidate rewriting, the expansion of the candidate rewriting is checked to see if it is contained in the expansion of the full MiCR. In Build-MaxCR, at the stage of finding the rewriting, the algorithm uses its strategy of constructing summary ACs to bypass any explicit containment checking between CQACs. However, the techniques described in this section can still be used at the later stage where the individual CQACs are being added to the UCQAC answer of Build-MaxCR. When Build-MaxCR is trying to add any CQAC $P'$ to its UCQAC answer $P$, it can use these techniques to check containment between CQAC $P'$ and each CQAC $P'_i$ that is already in $P$.

## 6 Experimental Results

In this section, we report the results of the experimental evaluation of our algorithms. Section 6.1 describes our experiments with the Build-MaxCR algorithm and Section 6.2 describes our results on the performance of the MiCR algorithms.

For our study, we used randomly generated queries of different shapes, such as chain queries and star queries [43]. We used the random query generator implemented in [11] to generate queries and views. This generator enables us to control the following parameters: (1) the number of subgoals in the queries and views; (2) the number of variables per subgoal; (3) the number of distinguished variables; and (4) the degree to which predicate names are duplicated in the queries and views. We extended the output from this generator so that the queries and views could have ACs in addition to relational subgoals. For this purpose we constructed ACs in which the two variables (or a variable and a constant) and the arithmetic operator in each AC was randomly selected. The algorithms were implemented in Java and compiled to executables. All experiments were run on a 2 GHz Pentium M processor running Windows XP Professional with 1 GB RAM and a 60GB hard drive. The run-times were averaged over twelve executions, after discarding the minimum and maximum readings.

### 6.1 Experiments on the Build-MaxCR algorithm

In this subsection, we discuss our experiments with the Build-MaxCR algorithm. Our results show the scalability and runtime of Build-MaxCR for chain and star queries.

Figures 1 and 2 show the runtime of Build-MaxCR for chain queries and views. Figure 2 shows that for $k = 2$, Build-MaxCR has good scalability — it can handle 600 views in about 10 seconds and 1200 views in about 30 seconds.

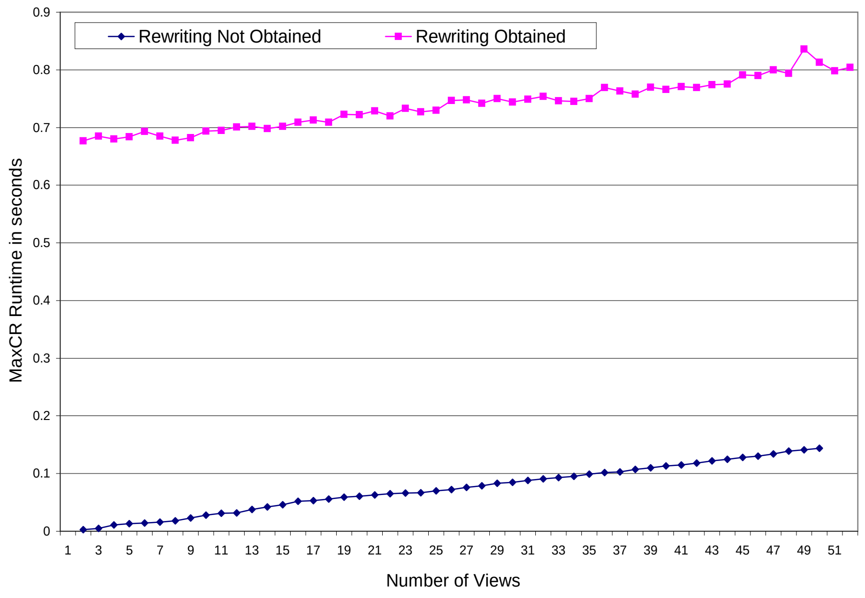

**Fig. 1.** Chain queries: 1 to 50 views.

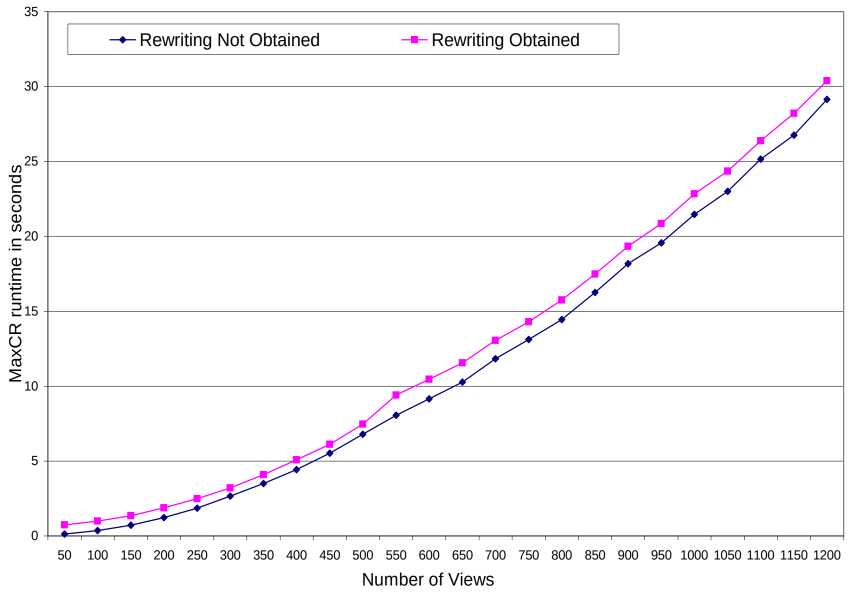

**Fig. 2.** Chain queries: Scalability up to 1200 views with k=2.

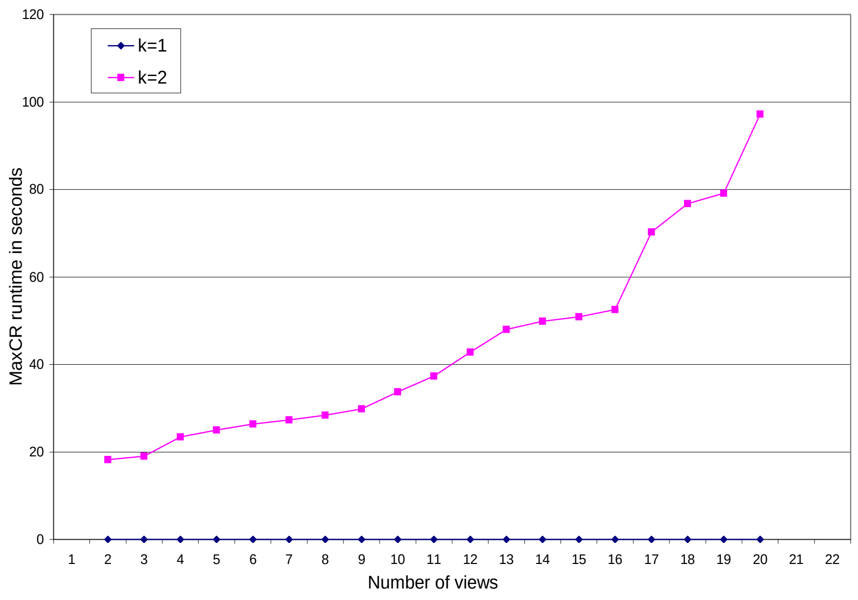

**Fig. 3.** Star queries.

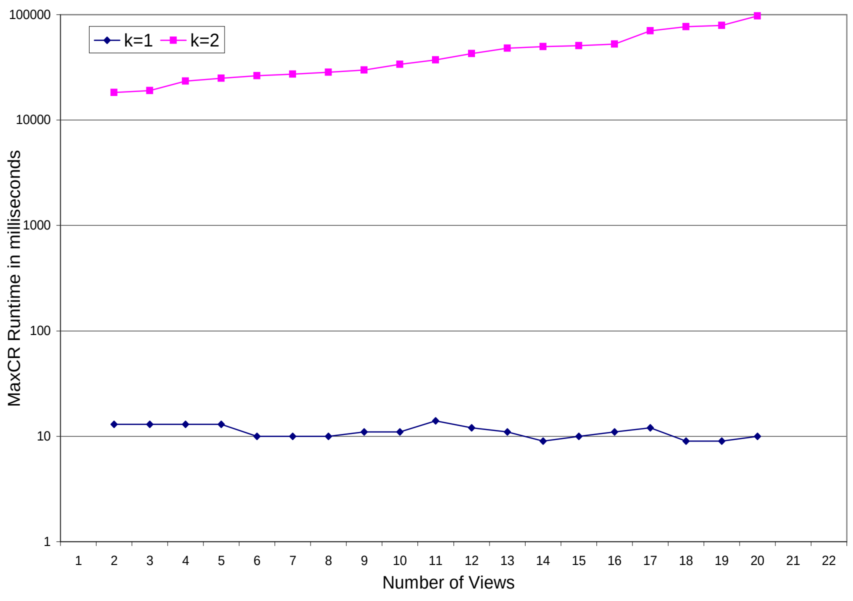

**Fig. 4.** Star Queries: Logarithmic Y-axis.

However, for higher values of $k$, as the size of the CQAC-rewriting templates becomes larger, the size of $MaxCrVars$ (the set of variables and constants used to form MaxCR canonical databases as explained in Section 3.2) also increases. For $|MaxCrVars| = n$ there are $O(n!)$ possible total orders, and this explosive growth in the number of total orders causes Build-MaxCR to have a correspondingly large runtime. However, for even small values of $k$, Build-MaxCR detects several multiple-mapping MCRs which cannot be detected by single-mapping algorithms. For example, the rewriting in Example 2 can be found by Build-MaxCR even for the low value of $k = 1$, however it can never be found using single-mapping algorithms.

Figures 1 and 2 show that for the same number of views and for the same value of $k$, Build-MaxCR always terminates earlier when the input query and views are such that there exists no contained rewriting of the given query using the given views. As shown in Figure 1, this saves at least 650 milliseconds for even the smallest input with just 2 views. These savings help lower the *average* runtime of Build-MaxCR, since in a real-world setting there may be a number of inputs for which no contained rewriting can be constructed using the given views. For example, if the query contains a subgoal with predicate name $p$, and if the CQAC-rewriting template being considered by Build-MaxCR does not contain any view which has a $p$-subgoal in its definition, then there cannot exist any mapping from the relational subgoals of $Q$ to the *same-name* subgoals in the expansion of the template. In such cases, Build-MaxCR immediately detects that $\mathcal{M} = \emptyset$ (as shown in the Build-MaxCR pseudocode in Algorithm 1 and does not do any further processing for this template. If the input viewset $\mathcal{V}$ is such that none of the views in it have a $p$-subgoal, then Build-MaxCR correctly detects that no contained rewriting is possible and terminates immediately without any further processing.

A single-mapping algorithm would suffice if queries were CQs (rather than CQACs), or if there were no non-distinguished variables. (If every variable in every view definition is available in the corresponding viewhead, then it becomes readily accessible to any algorithm which is trying to add ACs to a CQ-part in order to form a CQAC that is contained in the query.) However, in practice, it is not reasonable to expect such special conditions. Many real queries involve ACs. For example, 19 out of the 22 queries (i.e., more than 85% queries) in the TPC-H benchmark involve ACs. Similarly, it is also common to have views with non-head variables. The presence of shared, non-head variables gives rise to multiple-mapping contained rewritings. The Build-MaxCR algorithm can find such rewritings, even for low values of $k$.

Figure 3 shows the Build-MaxCR runtime for star queries and views, for $k = 1$ and $k = 2$. The runtime is almost constant (about 10ms) for $k = 1$. Figure 4 is an adaptation of Figure 3, with a logarithmic scale on the Y-axis. Similar to the case of chain queries, for star queries as well, our experiments show that for low values of $k$, the Build-MaxCR algorithm scales well.

## 6.2 Experiments on the MiCR algorithms

We performed experiments to compare the execution time of the pruned-MiCR algorithm with that of the CB-MiCR algorithm, and we report our results in this subsection. We measured the scalability of the two algorithms in the number of views. Not surprisingly, our results also show that it is common for the number of joins in a minimal MiCR to be significantly lower than the number of joins in a full MiCR.

Like the pruned-MiCR algorithm, the CB-MiCR algorithm too finds all possible $h(V)$'s by considering mappings from the views in $\mathcal{V}$ to the query $Q$.[18] After this stage, the pruned-MiCR algorithm uses its novel strategy to distribute the view subgoals into buckets, and then constructs a minimal MiCR by ensuring that each bucket is represented in the rewriting. It tries all subsets of the set of possible $h(V)$'s; candidate rewritings are formed by taking the conjunction of the subgoals in any such subset. The CB-MiCR algorithm considers the same candidate rewritings. However, in the absence of the MiCR-buckets it is forced to perform an expensive containment test for each candidate, to check if the expansion of the candidate is contained in the expansion of the full MiCR (i.e., in the conjunction of all $h(V)$'s). Our experiments demonstrate that the pruned-MiCR algorithm speeds up rewriting generation, since it eliminates containment checks by doing an early pruning in the process of generating a minimal MiCR. Thus the implementations of the two MiCR algorithms essentially differ in their strategies for testing candidates and make use of the same steps when possible.

In the first set of MiCR experiments, we studied the effect of increasing the number of views for chain queries. Figure 5 shows the results for a chain query with ten subgoals. It shows that the execution time of the CB-MiCR algorithm increases rapidly with an increase in coverage (i.e., the average number of view subgoals covering each query subgoal). Note that if each query subgoal is covered by at most one view, the pruned-MiCR algorithm's early pruning is not used. However, when there are multiple views covering a query subgoal, the advantages of the early pruning are substantial. The execution speedup resulting from the use of the pruned-MiCR algorithm is evident even at low coverage values of up to 2.

Figure 6 shows that the pruned-MiCR algorithm executes efficiently even for high coverage values. This is in sharp contrast to the CB-MiCR algorithm, which takes more than 20000ms even for coverages below 5. The increase in the run-time of the pruned-MiCR algorithm at higher coverage values is marginal and stems from the increased time required for forming the MiCR-buckets. In practice, constructing rewritings in the pruned-MiCR algorithm takes time that is about linear in the number of buckets. Thus the rewriting-construction phase does not significantly slow down the overall algorithm execution. The CB-MiCR algorithm however spends a substantial amount of time in rewriting construction, because for each candidate it has to do a CQAC containment test to check if the

[18] This is equivalent to chasing $Q$ with forward constraints obtained from the views in $\mathcal{V}$. Each homomorphism on a viewhead from $\mathcal{V}$ that gets added to the chase result of $Q$ is analogous to some $h(V)$ obtained by the pruned-MiCR algorithm.

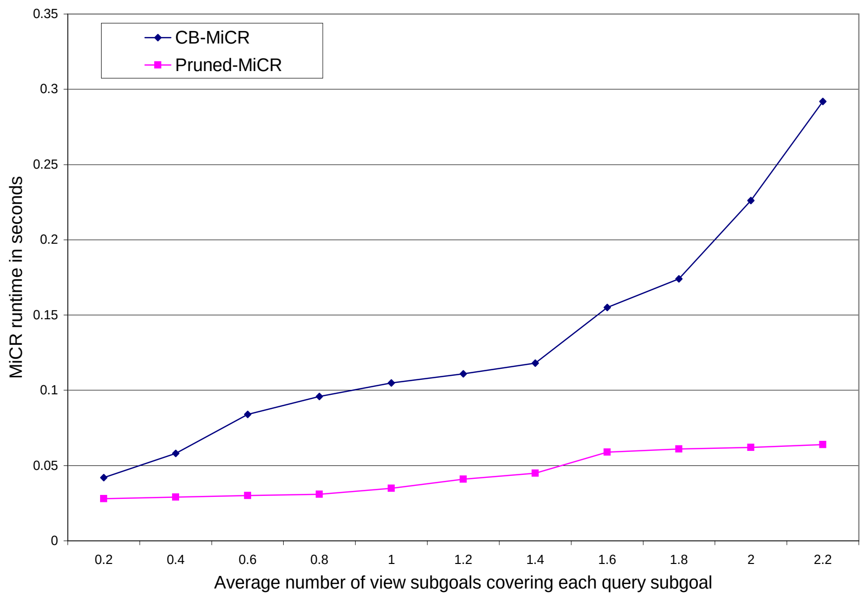

**Fig. 5.** Chain Queries: Coverage range 0 to 2.

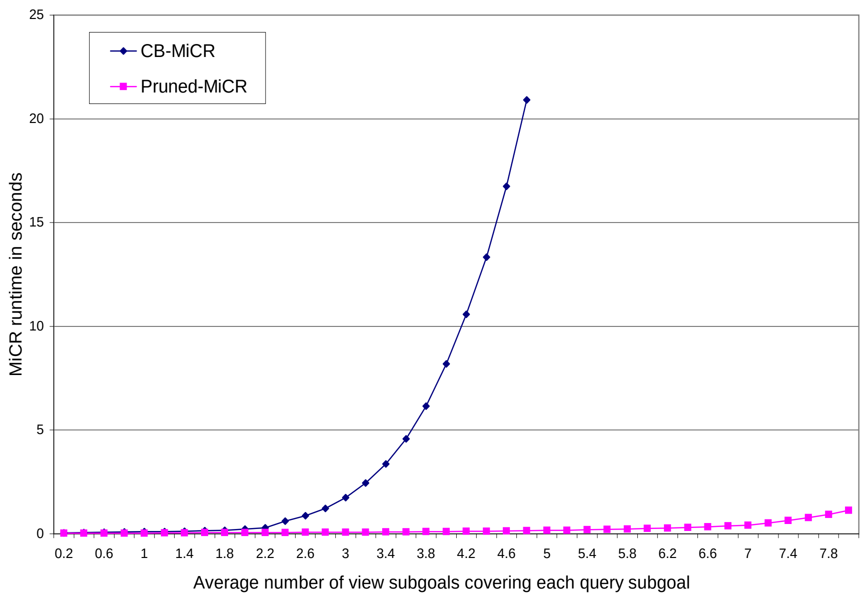

**Fig. 6.** Chain Queries: Coverage range 0 to 8.

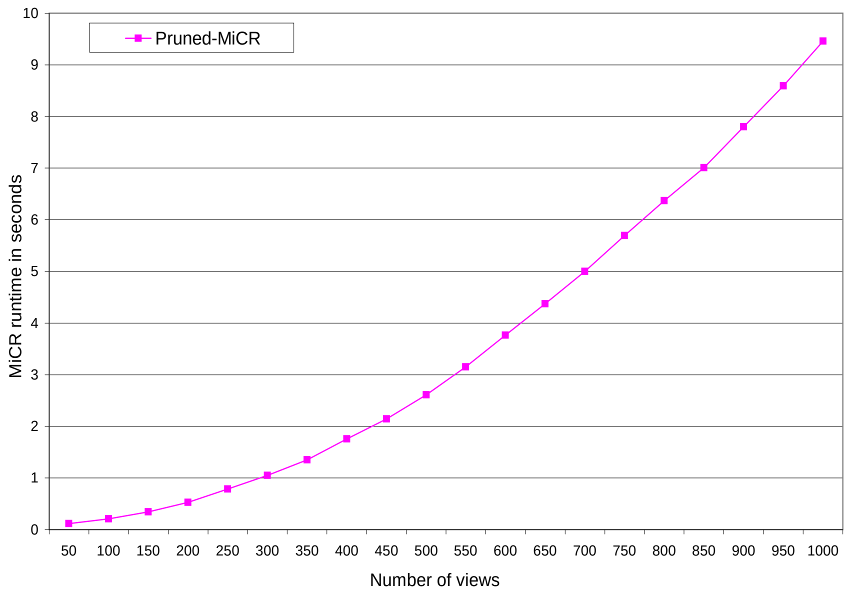

**Fig. 7.** Chain Queries: Scalability up to 1000 views.

candidate is contained in the full MiCR. If the full MiCR contains $m$ subgoals, then the CB-MiCR algorithm potentially has to perform $2^m$ such containment tests, as would be required if the full MiCR itself turns out to be the minimal MiCR. The scalability of the pruned-MiCR algorithm makes it useful in finding minimal MiCRs for practical cases involving a large number of applicable views. As seen in Figure 7, for a chain query with 10 subgoals and for chain views containing between 2 and 6 subgoals, the algorithm scales well and can handle even 1000 views in under 10 seconds.

Figures 8 and 9 show the runtime of CB-MiCR and pruned-MiCR on star queries and views. Figure 8 considers 1 to 10 views and Figure 9 considers up to 500 views. We used a query with one fact table that was joined with 5 dimension tables for a total of 6 relational subgoals. The views had between 2 and 8 relational subgoals.

Figure 10 shows the runtime of the pruned-MiCR algorithm for chain queries with and without the containment test which forms the last step of the algorithm. For coverage values of about 7 and below, the extra containment test does not appreciably slow down the algorithm.

In generating the MiCR of a query using views, every time that a new view is made available, it may generate 0 or more $h(V)$'s. And every time that a $h(V)$ is made available: (i) the number of joins in the MiCR increases by 1, to attain a new value of say $n$, and (ii) the number of joins in the minimal MiCR increases by 1, decreases by any amount, or remains the same, to take on some value between

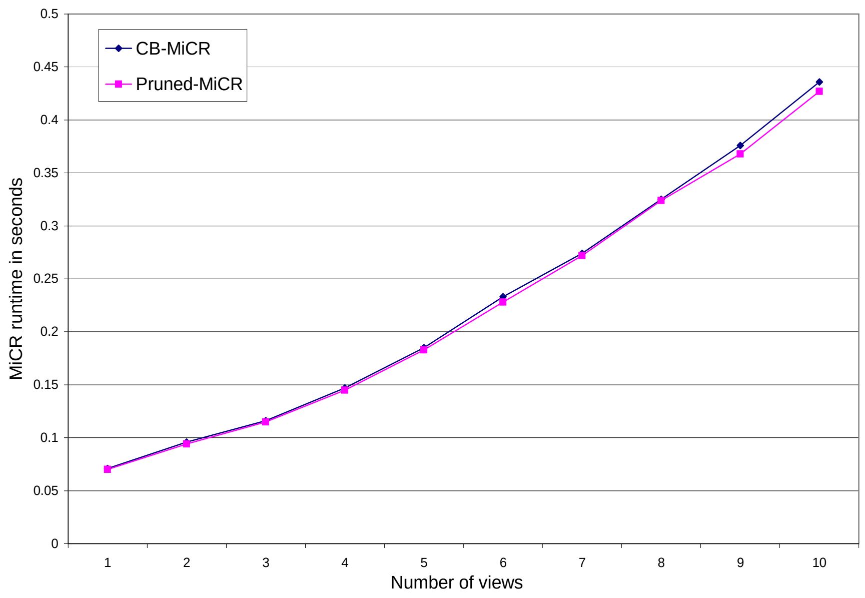

**Fig. 8.** Star Queries: 1 to 10 views.

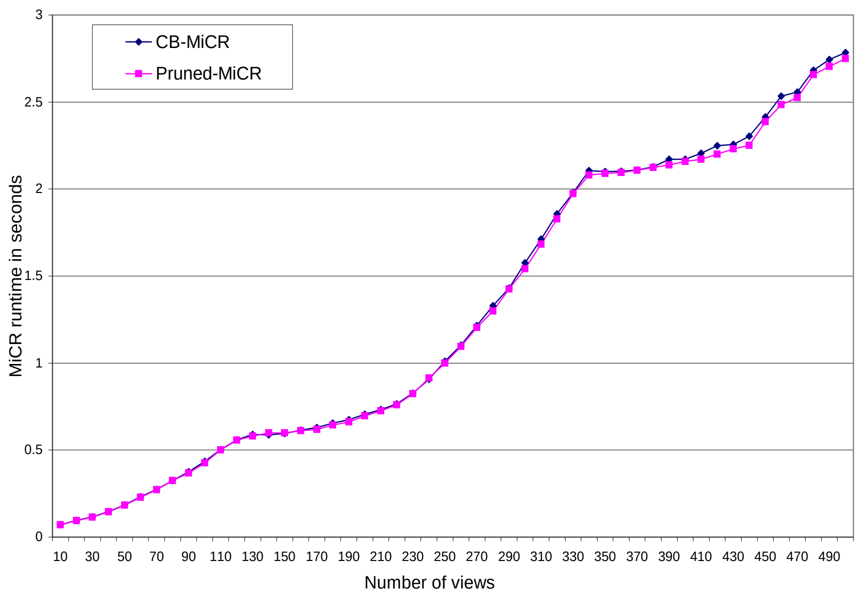

**Fig. 9.** Star Queries: Scalability up to 500 views.

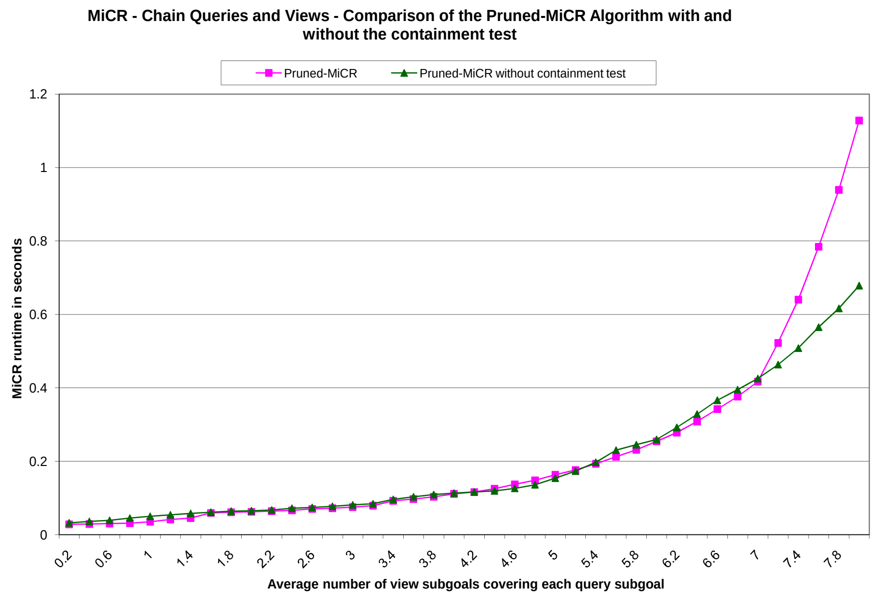

**Fig. 10.** Chain Queries: Runtime of the pruned-MiCR algorithm with and without the final containment test.

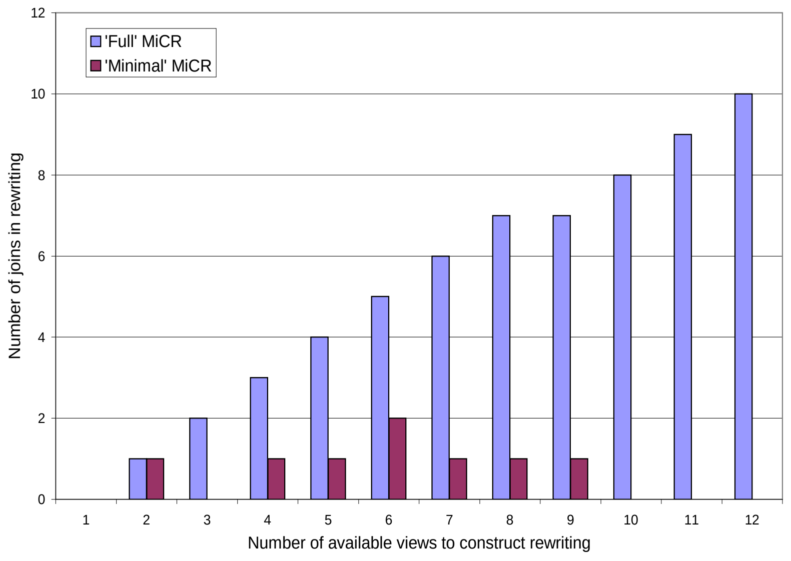

**Fig. 11.** Comparison of the number of joins in the full-MiCR and in the Minimal-MiCR.

| Number of available views | Query/Views | Number of joins | | Coverage |
|---|---|---|---|---|
| | | Full MiCR | Minimal MiCR | |
| | `Q() :- p(A, B), r(B, C), s(C, D), t(D, E)` | | | |
| 1 | `v1() :- p(A, B)` | 0 | 0 | 0.25 |
| 2 | `v2() :- r(B, C)` | 1 | 1 | 0.50 |
| 3 | `v3() :- p(A, B), r(B, C)` | 2 | 0 | 1.00 |
| 4 | `v4() :- s(C, D)` | 3 | 1 | 1.25 |
| 5 | `v5() :- r(B, C), s(C, D)` | 4 | 1 | 1.75 |
| 6 | `v6() :- t(D, E)` | 5 | 2 | 2.00 |
| 7 | `v7() :- s(C, D), t(D, E)` | 6 | 1 | 2.50 |
| 8 | `v8() :- r(B, C), s(C, D), t(D, E)` | 7 | 1 | 3.25 |
| 9 | `v9() :- u(L, M)` | 7 | 1 | 3.25 |
| 10 | `v10() :- p(A, B), r(B, C), s(C, D), t(D, E)` | 8 | 0 | 4.25 |
| 11 | `v11() :- s(C, D), u(L, M)` | 9 | 0 | 4.50 |
| 12 | `v12() :- r(B, C), t(D, E)` | 10 | 0 | 5.00 |

**Fig. 12.** Example query and views.

0 and $n$. In the worst case, the maximum value that $n$ can take is one less than the sum, over all query subgoals, of the number of $h(V)$'s covering that subgoal. The number of joins, both in the full and in the minimal MiCR, depends upon the number of views in the input. In general, a plot of the number of joins versus the number of available views may take any shape subject to conditions (i) and (ii) above, and normally the number of joins in a minimal MiCR is significantly lower than the number of joins in the full MiCR. In particular, once all query subgoals have been covered, the number of joins in a minimal MiCR can only decrease or remain the same, whereas the number of joins in the full MiCR go on increasing with every new $h(V)$. Figure 11 illustrates these ideas for the simple example of Figure 12.

Each bucket constructed by the pruned-MiCR algorithm represents a single subgoal from the given query. However, the presence of conditions such as shared variables may require the minimization procedure to construct multiple-subgoal buckets as well, in order to ensure that the correct minimal MiCR is obtained on all inputs. Such an increase in the number of buckets would increase the complexity and run-time of the algorithm. As a tradeoff, the pruned-MiCR algorithm represents only the single-subgoal buckets and these are sufficient to find the minimal-MiCR of several commonly encountered queries. However, since the pruned-MiCR algorithm does not construct multiple-subgoal buckets, there may be certain inputs on which it may not be able to find the minimal MiCR. (However note that it always finds the full MiCR on all inputs.) To ensure that the

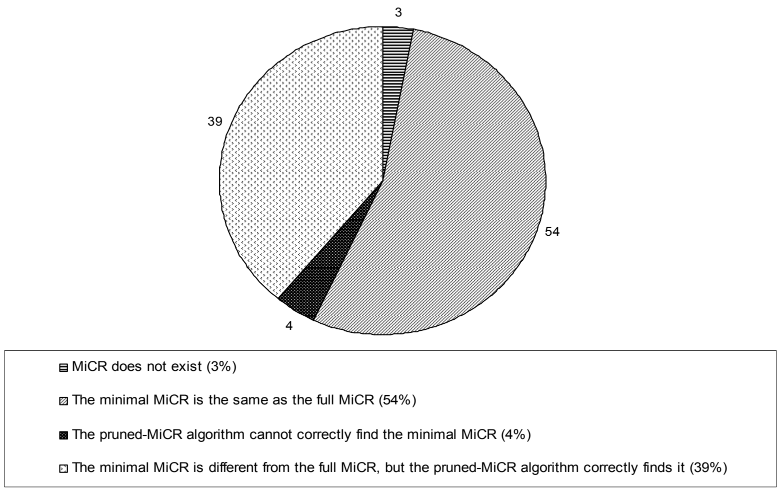

**Fig. 13.** The pruned-MiCR algorithm always finds the full MiCR. In addition, it correctly finds the minimal MiCR on 96% of the input sets.

minimized CQAC determined by the pruned-MiCR algorithm is equivalent to the full MiCR, the pruned-MiCR algorithm performs one containment check in its final step. We performed experiments to validate our claim that the buckets created by the pruned-MiCR algorithm can correctly find the minimized CQAC (i.e., the minimal MiCR, which is equivalent to the full MiCR) in most cases in practice. As shown in Figure 13, from a collection of over 1200 CQAC chain queries containing between one and ten relational subgoals, we created 100 sets of inputs, each consisting of one CQAC query and about 250 CQAC views. In only 4 of the 100 input sets, running the pruned-MiCR algorithm produced a minimized CQAC that was *not* equivalent to the full MiCR. In such cases the pruned-MiCR algorithm cannot find the minimal MiCR. Instead it outputs the full MiCR (which is the correct MiCR of the given query using the given views, except that it is not in the minimized form). Out of the remaining 96 cases, there were 3 cases in which there existed no MiCR of the given query using the given views (and this was correctly determined by the pruned-MiCR algorithm), 54 cases in which the minimal MiCR was the same as the full MiCR (and the pruned-MiCR algorithm was able to find it correctly), and 39 cases in which the minimal MiCR was different from the full MiCR (and in these cases too the pruned-MiCR algorithm was able to correctly find the minimal MiCR). Thus the choice of buckets that is made by the pruned-MiCR algorithm proved to be appropriate for finding the minimal MiCR in the case of over 95% of the input sets.

Thus in summary, our experiments have shown (i) that in addition to always finding the full MiCR, the pruned-MiCR algorithm also correctly finds the minimal MiCR on many of the inputs, (ii) that for chain queries, the pruned-MiCR algorithm outperforms the CB-MiCR algorithm significantly due to its early pruning, and (iii) that the pruned-MiCR algorithm scales gracefully with an increase in the number of views and is able to generate rewritings within a reasonable time for even a very large number of views.

## 7 Acknowledgments

The authors would like to thank Rachel Pottinger, for the random query generator which was helpful in generating inputs for the experiments.

## References

1. Bayardo, R., Bohrer, W., Brice, R., Cichocki, A., Fowler, J., Helal, A., Kashyap, V., Ksiezyk, T., Martin, G., Nodine, M., Rashid, M., Rusinkiewicz, M., Shea, R., Unnikrishnan, C., Unruh, A., Woelk, D.: InfoSleuth: Semantic integration of information in open and dynamic environments. In: SIGMOD. (1997) 195–206
2. Halevy, A.: Data integration: A status report. In: BTW. (2003) 24–29
3. Ullman, J.: Information integration using logical views. Theoretical Computer Science **239**(2) (2000) 189–210
4. Theodoratos, D., Sellis, T.: Data warehouse configuration. In: VLDB. (1997)
5. Chaudhuri, S., Krishnamurthy, R., Potamianos, S., Shim, K.: Optimizing queries with materialized views. In: ICDE. (1995) 190–200
6. Levy, A., Mendelzon, A., Sagiv, Y., Srivastava, D.: Answering queries using views. In: PODS. (1995) 95–104
7. Abiteboul, S., Duschka, O.: Complexity of answering queries using materialized views. In: PODS. (1998) 254–263
8. Afrati, F., Chirkova, R., Gergatsoulis, M., Pavlaki, V.: Finding equivalent rewritings in the presence of arithmetic comparisons. In: EDBT. (2006) 941–960
9. Halevy, A.: Answering queries using views: A survey. VLDB Journal **10(3)** (2001) 270–294
10. Mitra, P.: An algorithm for answering queries efficiently using views. In: Proceedings of the Australasian Database Conference. (2001)
11. Pottinger, R., Halevy, A.: MiniCon: A scalable algorithm for answering queries using views. VLDB Journal (2001)
12. Afrati, F., Li, C., Mitra, P.: Answering queries using views with arithmetic comparisons. In: PODS. (2002)
13. Tatarinov, I., Halevy, A.: Efficient query reformulation in peer data management systems. In: SIGMOD. (2004) 539–550
14. Halevy, A., Ives, Z., Madhavan, J., Mork, P., Suciu, D., Tatarinov, I.: The piazza peer data management system. IEEE Transactions on Knowledge and Data Engineering **16**(7) (2004) 787–798
15. Fuxman, A., Fazli, E., Miller, R.: Conquer: Efficient management of inconsistent databases. In: SIGMOD. (2005) 155–166
16. Fan, W., Chan, C., Garofalakis, M.: Secure XML querying with security views. In: SIGMOD. (2004) 587–598

17. Chandra, A., Merlin, P.: Optimal implementation of conjunctive queries in relational data bases. ACM STOC (1977) 77–90
18. Deutsch, A., Ludäscher, B., Nash, A.: Rewriting queries using views with access patterns under integrity constraints. In: ICDT. (2005) 352–367
19. Grahne, G., Mendelzon, A.: Tableau techniques for querying information sources through global schemas. In: ICDT. (1999) 332–347
20. Calvanese, D., Martinenghi, D., Cal, A.: Optimization of query plans in the presence of access limitations. In: ICDT 2007 Workshop on Emerging Research Opportunities in Web Data Management (EROW). Volume 229 of CEUR Electronic Workshop Proceedings, http://ceur-ws.org/Vol-229/. (2007)
21. van der Meyden, R.: The complexity of querying indefinite data about linearly ordered domains. In: PODS. (1992) 331–345
22. Klug, A.: On conjunctive queries containing inequalities. JACM **35**(1) (1988) 146–160
23. Afrati, F., Li, C., Mitra, P.: Rewriting queries using views in the presence of arithmetic comparisons. Theoretical Computer Science **368**(1-2) (2006) 88–123
24. Florescu, D., Levy, A., Suciu, D., Yagoub, K.: Optimization of run-time management of data intensive web-sites. In: VLDB. (1999) 627–638
25. Zaharioudakis, M., Cochrane, R., Lapis, G., Pirahesh, H., Urata, M.: Answering complex SQL queries using automatic summary tables. In: SIGMOD. (2000)
26. Levy, A., Rajaraman, A., Ordille, J.: Querying heterogeneous information sources using source descriptions. In: VLDB. (1996) 251–262
27. Afrati, F., Gergatsoulis, M., Kavalieros, T.: Answering queries using materialized views with disjunctions. In: ICDT. (1999) 435–452
28. Duschka, O., Genesereth, M.: Answering recursive queries using views. In: PODS. (1997) 109–116
29. Qian, X.: Query folding. In: ICDE. (1996) 48–55
30. Gupta, A., Sagiv, Y., Ullman, J., Widom, J.: Constraint checking with partial information. In: PODS. (1994) 45–55
31. Acharya, S., Gibbons, P., Poosala, V., Ramaswamy, S.: The Aqua approximate query answering system. In: SIGMOD. (1999) 574–576
32. Babcock, B., Chaudhuri, S., Das, G.: Dynamic sample selection for approximate query processing. In: SIGMOD. (2003) 539–550
33. Chakrabarti, K., Garofalakis, M., Rastogi, R., Shim, K.: Approximate query processing using wavelets. In: VLDB. (2000) 111–122
34. Poosala, V., Ganti, V., Ioannidis, Y.: Approximate query answering using histograms. IEEE Data Engineering Bulletin **22**(4) (1999) 5–14
35. Lee, A., Koeller, A., Nica, A., Rundensteiner, E.: Non-equivalent query rewritings. In: International Database Conference, Hong Kong. (July 1999)
36. Rizvi, S., Mendelzon, A., Sudarshan, S., Roy, P.: Extending query rewriting techniques for fine-grained access control. In: SIGMOD. (2004) 551–562
37. Miklau, G., Suciu, D.: A formal analysis of information disclosure in data exchange. In: SIGMOD. (2004) 575–586
38. Miklau, G.: Confidentiality and Integrity in Data Exchange. PhD thesis, University of Washington (2005)
39. Calvanese, D., Giacomo, G., Lenzerini, M., Vardi, M.: View-based query processing: On the relationship between rewriting, answering and losslessness. In: ICDT. (2005) 321–326
40. Afrati, F., Li, C., Mitra, P.: On containment of conjunctive queries with arithmetic comparisons (extended version). UCI ICS Technical Report (June 2003)

41. Afrati, F., Li, C., Ullman, J.: Generating efficient plans for queries using views. In: SIGMOD. (2001)
42. Afrati, F., Li, C., Mitra, P.: On containment of conjunctive queries with arithmetic comparisons. In: EDBT. (2004)
43. Steinbrunn, M., Moerkotte, G., Kemper, A.: Heuristic and randomized optimization for the join ordering problem. The VLDB Journal **6**(3) (1997) 191–208