ABSTRACT

CHANDRACHUD, MANIK. View-Based Rewriting Algorithms for Conjunctive Queries with Arithmetic Comparisons. (Under the direction of Professor Rada Chirkova.)

View-based query rewriting has important practical applications in a variety of data-management problems such as data integration, query optimization, data warehousing, and website design. Given a query and a set of views over a database schema, it is often necessary to rewrite the query in terms of the views. Answers to the query are then obtained by executing the rewriting. However, there may not always exist an equivalent rewriting of the given query using the given set of views. In this thesis, we study contained and containing rewritings; these rewritings produce respectively a subset and a superset of the set of query answers. In a given language, these represent best-approximation rewritings, as they provide the greatest underestimate and the least overestimate of the answer, respectively. We consider queries and views in the language of conjunctive queries with arithmetic comparisons (CQACs). The well-understood language of conjunctive queries (CQs) does not capture the in- or non-equalities that are characteristic of SQL select-project-join (SPJ) queries. Hence CQACs, which capture the full expressive power of SQL SPJ queries, are practically important to the many applications that need to use arithmetic comparisons.

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 union-of-CQAC MCR may have an unbounded number of CQAC disjuncts.) To the best of our knowledge, Build-MaxCR is the first algorithm to find union-of-CQAC MCRs for arbitrary CQAC queries and views. 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 algorithm pruned-MiCR may be very efficient to execute. Our experimental results indicate that our algorithms have good scalability and perform well in many practical cases, due to their extensive pruning of the search space.
View-Based Rewriting Algorithms for Conjunctive Queries with Arithmetic Comparisons

by

Manik Chandrachud

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Computer Science

Raleigh, North Carolina

2009

APPROVED BY:

Dr. Fereidoon Sadri
Dr. Alan Tharp

Dr. Rada Chirkova
Dr. Jon Doyle
Chair of Advisory Committee
BIOGRAPHY

Manik Chandrachud is currently a PhD student in the Computer Science Department at the North Carolina State University, Raleigh, NC.
ACKNOWLEDGMENTS

Some people enter graduate school reading advice articles on how to be an efficient researcher, how to communicate effectively, how to manage your time. While a few are lucky to have first-hand interaction with an advisor who actually embodies all of these rules of becoming a good academic. I am among the latter and am extremely fortunate to have had Dr. Rada Chirkova as my advisor. Numerous times and in a variety of situations I have admired her clarity of thought and lightning-quick ability to pick out what really matters as separate from what does not. I have benefited as much from her acute mathematical brain as I have from her encouraging words — in her prompt, unfailing appreciation of even the smallest well-done job. Dr. Chirkova is known for her deep, genuine interest in the wellbeing of her students. In my case, I have seen her go out of the way to make things even better, simply because she cares, if I may say so, in a strongly maternal way.

Dr. Prasenjit Mitra has been my co-advisor in collaborating with us on all the work presented in this thesis. His keen, intuitive grasp of subtle technical points has made every conversation with him an absolute pleasure. He has been a source of sound advice and many intellectually fulfilling discussions. But above all these, he has always been what I thought of as a really nice person, and I truly cherish the reassurance that this gave me throughout my graduate-student life. Dr. Foto Afrati has been my coauthor, and I gratefully acknowledge her participation and advice in shaping this work. I would also like to thank Dr. Rachel Pottinger, for the random query generator which was helpful in generating inputs for the experiments described in this work.

I am very grateful to Dr. Jon Doyle, Dr. Fred Sadri, and Dr. Alan Tharp for being on my thesis advisory committee. In my first year of graduate school, I had signed up for every class that Dr. Doyle was teaching. I would thoroughly enjoy attending his class lectures, and sometimes still miss being able to have that kind of three-times-a-week interaction with him. Our weekly reading-group meetings would never have been the same without Dr. Sadri’s invaluable comments and advice. I have learnt a great deal from his participation in those meetings, and have always benefited from his careful feedback on my work and my presentations. I have immensely enjoyed taking Dr. Tharp’s class. He always encouraged student interaction, and the lively class discussions that we used to have, remain one of the fondest, most memorable highlights of my time spent here.
I am thankful to our Director of Graduate Programs Dr. David Thuente for always being so extremely knowledgeable and cheerfully willing to advise me from time to time during the course of my graduate studies. Even after a brief five-minute visit, I have always left his office with a clearer, renewed perspective on my progress.

I would like to thank the entire past and current staff at the Computer Science Department, especially Carol Allen, Vilma Berg, Dare Cook, Ron Hartis, Linda Honeycutt, Ann Hunt, Margery Page, and Susan Peaslee, for their help with several administrative details and related matters. My thanks also to all my past and current officemates and students in the database reading group including Dongfeng Chen, Gang Gou, Maxim Kormilitsin, and Zohreh Asgharzadeh Talebi, for all the interesting discussions on research and otherwise. I would also like to express my gratitude to all my teachers for their contributions to my education so far. My heartfelt thanks to the many friends who have been such a wonderful part of my life — those whom I have known during my life here in graduate school, and all those whom I have known from before.

My family — my father, mother, her parents, and my grandfather Thatha — and the loving care they have always showered on me is way above anything that I can itemize to offer thanks on. So I simply express my deep affection for them, which they are already aware of. Thatha to me has been a living example on how to lead life, and its Real Goal, to which I dedicate this thesis.
# TABLE OF CONTENTS

**LIST OF TABLES** ........................................................................ vii

**LIST OF FIGURES** ..................................................................... viii

1 **Introduction** .................................................................................. 1

2 **Preliminaries** ................................................................................. 10
   2.1 Queries and Containment .............................................................. 10
      2.1.1 Containment of CQ queries ................................................. 12
      2.1.2 Containment of CQAC queries ........................................... 14
   2.2 Rewriting Queries Using Views .................................................. 15

3 **Contained Rewritings** ................................................................. 18
   3.1 Maximally Contained Rewritings ................................................ 18
      3.1.1 k-bounded Queries ............................................................. 20
   3.2 Preliminaries for Algorithm Build-MaxCR ............................... 22
   3.3 Description of Algorithm Build-MaxCR ................................... 26
   3.4 Examples Illustrating Algorithm Build-MaxCR ....................... 32
   3.5 Salient Features of Algorithm Build-MaxCR ......................... 38
   3.6 Possible Variations on the Steps of Build-MaxCR .................... 46
      3.6.1 Complexity of Algorithm Build-MaxCR ......................... 53
   3.7 Correctness of Algorithm Build-MaxCR .................................. 55
      3.7.1 Total-Order CQAC Queries .............................................. 55
      3.7.2 Soundness of Build-MaxCR ............................................. 60
      3.7.3 Completeness of Build-MaxCR ...................................... 62

4 **Containing Rewritings** ............................................................... 66
   4.1 Minimally Containing Rewritings ............................................. 67
   4.2 Preliminaries for the MiCR Algorithms .................................... 67
   4.3 Description of the MiCR Algorithms ....................................... 69
      4.3.1 The Algorithm Full-MiCR .................................................. 71
      4.3.2 The Algorithm Pruned-MiCR ............................................ 73
      4.3.3 The Algorithm CB-MiCR .................................................. 74
      4.3.4 Bucket Construction in Pruned-MiCR: Illustrative Example .. 75
      4.3.5 Incompleteness of Algorithm Pruned-MiCR ..................... 78
      4.3.6 Complexity of Algorithm Pruned-MiCR ......................... 84
   4.4 Correctness of the MiCR Algorithms ....................................... 84
   4.5 Variable Types in Arithmetic Comparisons ............................. 88
LIST OF TABLES

Table 1.1 Our contributions, previous work, and applications. ......................... 6

Table 3.1 Finding summary ACs for Build-MaxCR. ........................................ 40

Table 4.1 The minimal MiCR for the different $\mathcal{V}$’s in Example 4.5 (page 82). ....... 82

Table 4.2 The minimal MiCR for the different $\mathcal{V}$’s in Example 4.6 (page 82). .......... 83

Table 4.3 The minimal MiCR for the different $\mathcal{V}$’s in Example 4.7 (page 83). .......... 83
LIST OF FIGURES

Figure 2.1 Mapping and covering of CQ query subgoals in the context of contained rewritings.................................................................................................................. 13

Figure 5.1 Chain queries with 1 to 50 views: Build-MaxCR runtime........................................... 94

Figure 5.2 Chain queries with up to 1200 views: scalability of Build-MaxCR with k=2.................. 95

Figure 5.3 Star queries with 1 to 20 views: Build-MaxCR runtime.............................................. 95

Figure 5.4 Star queries: Build-MaxCR runtime on a logarithmic Y-axis........................................ 96

Figure 5.5 Chain queries with coverage range 0 to 2: full+pruned-MiCR and full+CB-MiCR runtimes........................................................................................................ 98

Figure 5.6 Chain queries with coverage range 0 to 8: full+pruned-MiCR and full+CB-MiCR runtimes........................................................................................................ 99

Figure 5.7 Chain queries with up to 1000 views: scalability of algorithm full+pruned-MiCR ......... 99

Figure 5.8 Star queries with 1 to 10 views: full+pruned-MiCR and full+CB-MiCR runtimes........... 100

Figure 5.9 Star queries with up to 500 views: scalability of the MiCR algorithms......................... 101

Figure 5.10 Chain queries: runtime of algorithm full+pruned-MiCR with and without the final containment test................................................................. 102

Figure 5.11 Comparison of the number of joins in the full MiCR and in the minimal MiCR......... 102

Figure 5.12 Example query and views for the graph in Figure 5.11. ............................................. 103

Figure 5.13 Chain queries with 250 views: Algorithm full+pruned-MiCR always finds the full MiCR. In addition, it correctly finds the minimal MiCR on 96% of the input sets................................................. 104

Figure 5.14 Chain queries with 300 views: Algorithm full+pruned-MiCR always finds the full MiCR. In addition, it correctly finds the minimal MiCR on 93% of the input sets................. 104
Figure 5.15 Chain queries with 100 views: Algorithm full+pruned-MiCR always finds the full MiCR. In addition, it correctly finds the minimal MiCR on 100% of the input sets. 105

Figure 5.16 Chain queries with 150 views: Algorithm full+pruned-MiCR always finds the full MiCR. In addition, it correctly finds the minimal MiCR on 99% of the input sets. 105

Figure 5.17 Chain queries with 200 views: Algorithm full+pruned-MiCR always finds the full MiCR. In addition, it correctly finds the minimal MiCR on 94% of the input sets. 106

Figure 5.18 Full+pruned-MiCR with chain queries: distribution of the types of result on the 100 input sets. 107

Figure 5.19 Full+pruned-MiCR consistently finds the correct answer on the full and the minimal MiCR in over 90% of the cases at all times. 107

Figure 5.20 Full+pruned-MiCR with chain queries: variations in the types of outcomes for different numbers of views in the input set. 108
Chapter 1

Introduction

In the context of a database management system, a query (in a given database-querying language) is a statement that can be used to retrieve user-specified data from the database. It can also be used to create, modify, or delete the data, or the database schema itself. Instead of directly executing a query $Q$ over the database, it may be possible to compute the answer to $Q$ by making use of derived data called views. A view $V$ is a named query, which can be stored in a database system either as a definition of $V$ (virtual view) or as a relation obtained by running $V$ over the database (materialized view). A user query $Q$ can be answered using views such as $V$ via a new definition called a rewriting, which is a query expressed in terms of the views. Given a query $Q$ and a set of views $\mathcal{V} = \{V_1, V_2, \ldots, V_n\}$ over the same database schema, suppose that $R$ is a rewriting of $Q$ that is formulated in terms of only the views in $\mathcal{V}$. If $R$ is an equivalent rewriting of $Q$, then the set of answers produced by executing $R$ is identical to the set of answers produced by executing $Q$. At the same time, executing $R$ in place of $Q$ could be much more efficient if, for example, part of the required computation has already been performed while materializing $\mathcal{V}$. As another example, in some security applications, a user’s access to the database may be defined only through the viewset $\mathcal{V}$, that is, the user may not have direct access to the database tables at all. Hence, in order to obtain information from the database, the user has no choice but to rewrite his query using only the views from $\mathcal{V}$. In general, rewriting queries using views is an important technique and is used in a variety of data-management

---

1The schema is the structure of the database. In a relational database, the definition of the tables, the fields in the tables, the relationships between the fields and the tables are all considered to be a part of the schema.
problems \cite{Hal01} such as data integration \cite{Hal03, Ull00, LRO96, BBB+97, DG97}, query optimization \cite{CKPS95, LMSS95}, answering queries using views \cite{LMSS95}, maintaining physical data independence \cite{PDST00, YL87}, and data warehousing and website design \cite{HRU96, TS97, FLSY99}.

Previous research has focused primarily on obtaining equivalent rewritings of queries, that is rewritings that can be used to derive exact query answers (see, e.g., \cite{LMSS95, AD98, ACGP06, ALU07}). When equivalent rewritings cannot be found, then in many applications it makes sense to work with contained rewritings, which return a subset of the set of the query answers. Of especial interest in this context are so-called maximally contained rewritings (MCRs), which can be used to obtain a maximal subset of the query answers that can be obtained using the given views (see, e.g., \cite{Hal01, Ull00, Mit01, PH01, ALM02}). In addition, in applications such as querying the World-Wide Web, mass marketing, searching for clues related to terrorism suspects, or peer data-management systems (see, e.g., \cite{TH04, HIM+04}), users prefer to get a superset of the query answers, rather than getting no answers at all (when no equivalent or contained rewritings exist). In such scenarios, users might be interested in containing rewritings, which return a superset of the set of the query answers. Minimally containing rewritings (MiCRs) \cite{DLN05, GM99, CCM07} are the containing rewritings that return the fewest false positives when answering the query.

In this thesis, we study maximally contained and minimally containing rewritings of queries using views, which we refer to collectively as approximate rewritings. In a given language, these represent best-approximation rewritings, as they provide the user with the greatest underestimate and the least overestimate of the answer, respectively. We now present a motivating example for some of the types of rewritings that we consider in this thesis.

**EXAMPLE 1.1.** Consider the following SQL \cite{MS93} 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.

\[
Q: \text{SELECT AgentName, ContactPhone FROM HomesForSale WHERE Price < 500000 AND NumberOfBedrooms } \geq 3;
\]

Suppose that $\mathcal{V} = \{V_1, V_2, V_3, V_4, V_5, V_6\}$ is the set of views (with straightforward SQL definitions) that can be used to produce rewritings of $Q$. We assume that the language of SQL select-project-join (SPJ) queries is being used to construct rewritings of $Q$ using $\mathcal{V}$. 
A rewriting that uses \( V_1 \) alone\(^2\) is a contained rewriting of \( Q \) using \( \mathcal{V} \), since on some databases it may return a proper subset of the query answer, rather than the exact answer. Similarly, each of \( V_2 \) and \( V_3 \) is also a contained rewriting. On the other hand, each of \( V_4 \) and \( V_5 \) is a containing rewriting of \( Q \) using \( \mathcal{V} \), as it may return either the exact query answer or a proper superset of the query answer.

Although each of \( V_1, V_2, \) and \( V_3 \) is a contained rewriting of query \( Q \), only \( V_1 \) and \( V_3 \) are maximally contained rewritings (MCRs) of \( Q \) using \( \mathcal{V} \) in our SQL SPJ rewriting language. In this language, \( V_2 \) is not an MCR because on all databases its answers are a subset of the set of answers to rewriting \( V_1 \). If we consider a more expressive rewriting language, which allows us to take union\(^3\) of SQL SPJ queries, the union of \( V_1 \) and \( V_3 \) would be the MCR of \( Q \) using \( \mathcal{V} \), while \( V_1 \) or \( V_3 \) alone would no longer be MCRs. Similarly, among containing rewritings, the intersection of \( V_4 \) and \( V_5 \) is a “more minimally containing” rewriting of \( Q \) as compared to \( V_4 \) (or \( V_5 \)) alone. This is because the set of answers obtained for the intersection of \( V_4 \) and \( V_5 \) is a subset of the set of answers obtained for \( V_4 \) (or \( V_5 \)) alone.

Also, note that although \( V_6 \) is very similar to \( V_5 \), it does not form a containing rewriting,

\[
V_1: \text{SELECT AgentName, ContactPhone FROM HomesForSale}
\text{WHERE Price < 300000 AND NumberOfBedrooms >= 3;}
\]

\[
V_2: \text{SELECT AgentName, ContactPhone FROM HomesForSale}
\text{WHERE Price < 250000 AND NumberOfBedrooms >= 3;}
\]

\[
V_3: \text{SELECT AgentName, ContactPhone FROM HomesForSale}
\text{WHERE Price > 400000 AND Price < 450000 AND NumberOfBedrooms >= 3;}
\]

\[
V_4: \text{SELECT AgentName, ContactPhone FROM HomesForSale}
\text{WHERE Price < 600000;}
\]

\[
V_5: \text{SELECT AgentName, ContactPhone FROM HomesForSale}
\text{WHERE NumberOfBedrooms >= 2;}
\]

\[
V_6: \text{SELECT ContactPhone FROM HomesForSale}
\text{WHERE NumberOfBedrooms >= 2;}
\]
because it does not select the attribute \texttt{AgentName} and hence a rewriting consisting of V6 alone would not be safe\footnote{If we try to construct a query which is a rewriting of Q using V6 alone, then the attribute \texttt{AgentName} required to answer Q is not available to us through V6, which is the only view used in the rewriting. In the context of database queries, such a query is said to be “unsafe”. Please see Section \ref{sec:rewritings} on page \pageref{sec:rewritings} for a definition.}

Suppose this database is hosted by a real-estate agency that does not mind all of its \texttt{HomesForSale} being visible in response to user searches on its website, except for homes costing more than $2,000,000, for which it expects users to call its office directly. In such a case, if both V5 and V6 were modified to include \texttt{price} $\leq 2000000$ in their \texttt{WHERE} clause, then the set $\mathcal{V}$ would become a \textit{secure} viewset, which does not allow users to access any data that the agency does not wish to disclose. This is accomplished because for any query that seeks information on homes that are priced at $2,000,000$ or more, there is no way to produce a rewriting for such a query using only the views in $\mathcal{V}$.

In our work we focus on conjunctive queries\footnote{A conjunctive query is basically a single non-recursive datalog \cite{Ull89} rule, please see Section \ref{sec:rewritings} on page \pageref{sec:rewritings} for details.} with arithmetic comparisons (CQACs), that is on the language capturing the full expressive power of practically important SQL \cite{MS93} select-project-join (SPJ) queries. (The well-understood language of conjunctive queries (CQs) \cite{CM77} 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). The well-studied (for CQ queries and views) problems of finding equivalent rewritings and MCRs are recognized as being significantly more complex for CQAC queries, with many practically important cases still unexplored \cite{Ull00,ALM02}. The complexity of the problems in the presence of arithmetic comparisons (ACs) is mainly due to the more complex containment test — the containment test is NP-complete in the case of CQ queries \cite{CM77} but $\Pi_2^P$-complete \cite{Ull00,vdM92} in the case of CQAC queries.

The specific contributions presented in this thesis are as follows:

- **Contained rewritings:** Pottinger and Halevy developed algorithm MiniCon IP \cite{PH01}, which efficiently finds UCQAC MCRs for special cases of CQAC queries, views, and rewritings, specifically for those cases where the “homomorphism property” \cite{Klu88} holds between the expansions of the rewritings and the query\footnote{Intuitively, the \textit{homomorphism property} is said to hold between CQAC queries $Q_1$ and $Q_2$ when a single}. At the same
time, MiniCon IP cannot find the rewriting for the general case. (For example, it cannot find the rewriting in Example 3.5 on page 34.) We present a sound and complete algorithm called \textit{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 3.5, Build-MaxCR returns the rewriting $P'$ of the example.) The size-limit restriction of Build-MaxCR is due to the fact that for CQAC queries and views, a view-based UCQAC MCR may have an unbounded number of CQAC disjuncts, see Example 3.1 on page 19. To the best of our knowledge, Build-MaxCR is the first algorithm to find UCQAC MCRs for arbitrary CQAC queries and views.

- **Containing rewritings:** 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 expensive join operations in the evaluation plans for the MiCR. Our main contribution is a sound and efficient algorithm that we call \textit{pruned-MiCR}. Given a CQAC MiCR for a given problem input (CQAC query and views), pruned-MiCR does \textit{global} minimization of the MiCR, and in many cases produces MiCR formulations whose evaluation costs are significantly lower than those of the (MiCR) input to the algorithm. The idea of pruned-MiCR is quite general and is thus applicable beyond containing rewritings. Specifically, a straightforward modification of pruned-MiCR could be used to reduce the number of joins of (and thus to provide more efficient execution options for) the outputs of our algorithm Build-MaxCR. Algorithm pruned-MiCR is always sound. However, in cases where the homomorphism property (between the query and the expansion of the rewriting) does not hold, algorithm pruned-MiCR may not be complete. Hence we also present another MiCR algorithm called the \textit{CB-MiCR algorithm}. CB-MiCR has the same inputs and outputs as pruned-MiCR, but uses an exhaustive search strategy and is thus sound and complete for all inputs.

- **Reducing the runtime of containment checking:** We study the problem of reducing the runtime of containment checking between two CQAC queries, and propose mapping (i.e., a correspondence) from $Q_1$ to $Q_2$ is sufficient to establish the containment of $Q_2$ in $Q_1$; please see Definition 3.9 on page 25 in Section 3.2 for the formal definition.
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 the proposed algorithms Build-MaxCR and pruned-MiCR.

- **Formal framework for total-order CQAC queries:** We give the proofs of soundness and completeness for Build-MaxCR and for the MiCR algorithms. We present these theoretical results along with a number of detailed definitions, and establish a formal framework for “total-order CQAC queries”. This is a contribution of independent interest to any work that needs to consider all canonical databases (please see Section 2.1.2 on page 14 and Section 3.2 on page 22 for the definition) of a CQAC query. In particular, in our work on MCRs, it sets up the foundation needed to prove the correctness of Build-MaxCR.

- **Implementation and experimental evaluation:** Based on our implementation of the Build-MaxCR, pruned-MiCR, and CB-MiCR algorithms, we present the results of our extensive experimental study of their performance and scalability on various types of queries. Our experimental results indicate the superior performance of pruned-MiCR in comparison to CB-MiCR.

- **Directions for future research:** We also point out some related open problems that would be promising directions for future research in this area.

<table>
<thead>
<tr>
<th>Algorithms we present</th>
<th>Contained Rewritings</th>
<th>Containing Rewritings</th>
</tr>
</thead>
<tbody>
<tr>
<td>UCQAC size-limited MCR for CQACs [ACCM09]</td>
<td>Global minimization of MiCR for CQACs [ACCM09]</td>
<td></td>
</tr>
<tr>
<td>Previous work</td>
<td>MCR [Hal01, PL00, Mit01, ALM02]</td>
<td>MiCR [GM99, DLN05]</td>
</tr>
<tr>
<td>Applications</td>
<td>Data warehousing, security, privacy</td>
<td>Mass marketing, P2P, information retrieval</td>
</tr>
</tbody>
</table>

Table 1.1 gives a summary of our results and contributions. While the running-time complexity of our proposed algorithms is high in the worst case (doubly exponential
for algorithm Build-MaxCR, and singly exponential for algorithm pruned-MiCR), 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.

In the remainder of this chapter, we present a brief discussion of existing research that is related to the work in this thesis. The rest of the thesis is organized as follows. Chapter 2 summarizes some preliminaries on view-based query rewriting and introduces the notation that is used throughout the thesis. Chapter 3 deals with contained rewritings. It formulates the theoretical basis for algorithm Build-MaxCR, then describes the algorithm itself through pseudocode and examples, outlines some salient features and possible variations of Build-MaxCR, and finally gives proofs for its soundness and completeness. Chapter 4 presents algorithms pruned-MiCR and CB-MiCR for finding efficient to execute formulations of CQAC minimally containing rewritings of CQAC queries. The details of the experimental evaluations of all three algorithms are included in Chapter 5. Finally, we conclude in Chapter 6.

Related Work

The problem of using views in query answering [LMSS95] is relevant to applications in information integration [Ull00], data warehousing [Hal01], website design [FLSY99], and query optimization [CKPS93, LMSS95, ZCL00]. Algorithms for finding rewritings of queries using views include the bucket algorithm [GM99, LRO96], the inverse-rules algorithm [AGK99, DG97, Qia96], the MiniCon algorithm [PH01], and the shared-variable-bucket algorithm [Mit01]; see [Hal01] for a survey. Almost all of the above work focuses on investigating MCRs or equivalent rewritings [Ull00, AD98], 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 [GSUW94, Kin88, vdM92].

Since we consider rewritings that may return false positives or false negatives, our work has similarities with work on approximate answering of queries using views, see [AGPR99, BCD03, CGRS00, PGI99] and references therein. Approximate query answering is useful when exact query answers cannot be found, and the user would rather have
Lee et al. [LKR99] 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 was introduced in [GM99] and also addressed in [DLN05, CCM07]. Deutsch and colleagues [DLN05] provided approaches for solving the problem of rewriting queries using views with limited access patterns [RSU95] under integrity constraints, and proved that feasibility is NP-complete for queries, views, and constraints over unions of conjunctive queries (UCQ queries) and $\Pi_2^p$-complete for UCQ queries with negation. These results hold in those cases where the chase [DNR08] terminates and its result is not “too large.” They also presented an algorithm that is guaranteed to find an equivalent rewriting (if one exists) or at least the minimally containing rewriting (unique if it exists). In addition, they gave an algorithm for finding the maximally contained rewriting for UCQ queries with negation, and extended the above results to handle equality and arithmetic comparisons by modeling them with integrity constraints. All of the algorithms of [DLN05] are applicable only in those cases where chase [DNR08] on the applicable problem inputs terminates.

[ACGP06] presented a sound and complete algorithm that returns a UCQAC equivalent rewriting of the input CQAC query in terms of the input CQAC views. In this thesis, we focus on those problem settings where one is to find a rewriting of a given CQAC query in terms of given CQAC views, but an equivalent UCQAC rewriting does not exist, and thus the algorithm of [ACGP06] returns no answer. Specifically, the algorithm of [ACGP06] returns no answer when given the query/view combinations of the motivating and illustrative examples of this thesis, namely Example 3.1 (page 19), Example 3.5 (page 34), and Example 4.3 (page 75).

[ALM06] solved the problem of finding the UCQAC MCR for CQAC Q using CQAC views, for the limited case where the homomorphism property holds between the query and the expansion of the rewritings. Also, [PH01] presented MiniCon IP (an extension

---

7 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 (only) or false negatives (only).
of the MiniCon algorithm [PL00], which efficiently finds the MCRs of CQACs. However, as seen in Example 3.5 (page 34), if the homomorphism property does not hold, then MiniCon IP may not be able to find the MCR. In contrast, algorithm Build-MaxCR that we present works for arbitrary queries and views.

Other related work includes [CKPS95], which considered the problem of generating efficient plans using materialized views by replacing subgoals in a query with view literals. [GKC06] investigated the problem of finding efficient equivalent view-based rewritings of relational queries, possibly involving grouping and aggregation, and proposed sound algorithms that extend the cost-based query-optimization approach of System R [SAC+79]. [CNS06] considered the problem of rewriting aggregate queries with arbitrary aggregation functions. In [RMSR04], query-rewriting techniques were used for access control. [MS04, MS07] presented 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 [Mik05]. Calvanese et al. [CGLV05, CGLV07] discussed query answering, rewriting, and losslessness with respect to two-way regular path queries. In our work, we concentrate only on query rewritings for CQAC queries.
Chapter 2

Preliminaries

In this chapter, we review some standard concepts about answering queries using views, and summarize some results from the literature related to query containment. We also introduce some notation that we will use throughout the thesis.

2.1 Queries and Containment

In this section, we describe the different types of conjunctive queries and their respective expressive powers. We also present some theorems from the literature that are central to any work on query containment and rewriting.

We consider the language of conjunctive queries with arithmetic comparisons (CQACs). CQAC queries are SQL select-project-join queries with equality and arithmetic-comparison (i.e., in- or non-equality) selection conditions. A CQAC query can be represented as follows:

\[ h(\vec{X}) :– g_1(\vec{X}_1), ..., g_n(\vec{X}_n), C_1, ..., C_m. \]

\( h(\vec{X}) \) is the head of the query, which represents the answer, that is, the result of running the query. The remaining part of the query after the “:\=” sign is the query body, which consists of a number of subgoals (also called atoms) connected by “,”, “&”, or “\&” to represent their conjunction or logical ANDing. The body is made up of a set of relational subgoals \( g_1(\vec{X}_1), ..., g_n(\vec{X}_n) \), and a set of arithmetic subgoals \( C_1, ..., C_m \). The relational subgoals are also called regular subgoals, ordinary subgoals, or uninterpreted subgoals. The arithmetic subgoals are also called arithmetic predicates, comparison predicates, interpreted predicates,
or arithmetic comparisons (ACs). For a CQAC query $Q$, the relational part is sometimes denoted by $\text{core}(Q)$ or $Q_0$ while the AC part is denoted by $AC(Q)$ or $\beta_Q$. $\text{head}(Q)$ denotes the head of $Q$.

Each relational subgoal $g_i(\bar{X}_i)$ has a relation-name $g_i$ and a tuple of arguments $\bar{X}_i$ corresponding to the relational schema of $g_i$. Each argument in $\bar{X}_i$ is either a variable or a constant. The variables $\bar{X}$ that occur in the head of the CQAC query are called head variables, or distinguished variables. All other variables in the CQAC query are called nonhead, nondistinguished, or existential variables. A variable which occurs in more than one subgoal in the body of the query is called a shared variable.

Each AC $C_i$ has the form $S_1 \theta S_2$, where $S_1$ and $S_2$ are variables or constants, and the operator $\theta$ is one of $\{<,>,\leq,\geq,=,\neq\}$. If $\theta$ is $=$ then the AC is called an equality AC, otherwise it is called a non-equality AC. If $\theta$ is $\neq$ then the AC is called an in-equality AC. We assume that database instances are over densely totally ordered domains.

A CQ query is a special case of a CQAC query in which there are no ACs (except possibly equality ACs). The language of Unions of Conjunctive Queries with Arithmetic Comparisons (UCQACs) is more expressive than the language of CQACs, since a UCQAC query is a disjunction (i.e., a logical OR) of CQAC queries. Similarly, the language of Unions of Conjunctive Queries (UCQs) is more expressive than the language of CQs, since a UCQ query is a disjunction of CQ queries.

Note that there are two ways of expressing the equality between variables in a CQAC query — either implicitly by giving all of them the same name, or explicitly by adding equality ACs to equate them. A CQAC query is said to be expressed in the normalized or rectified form, if the head and relational subgoals of the query are completely free of repeated variables and constants. To obtain the normalized form of a query, any occurrences of constants or repeated occurrences of variables in it are eliminated by introducing fresh variables, and adding extra compensating ACs to equate the new variables to the original constants or variables. For a CQAC query $Q$, it is possible to take the closure of $AC(Q)$ by conjoining with the ACs that are already in $AC(Q)$, all possible ACs that can be logically derived from $AC(Q)$. All the queries we consider are safe, that is each distinguished variable or variable appearing in the $\beta$ of the query also appears in at least one relational subgoal of the query. We use the term semi-interval CQAC (SI-CQAC) query to refer to conjunctive queries with arithmetic comparisons, where each comparison in the query is either one of
A nondistinguished variable $X$ in a CQAC query $Q$ is said to be an exportable variable or a distinguishable variable if there are two or more distinguished variables $X_1, X_2, \ldots, X_n$ in $Q$, such that the equation $X_1 = X_2 = \ldots = X_n$ and the ACs in $\text{closure}(AC(Q))$ together imply that $X = X_1 = X_2 = \ldots = X_n$. In this case, we say that variable $X$ can be exported by equating variables $X_1, X_2, \ldots, X_n$.

To find the exportable nondistinguished variables in a CQAC query $Q$, we use the ACs in $\text{closure}(AC(Q))$ to construct the inequality graph $[Klu88]$ for $Q$, denoted $G(Q)$. That is, for each AC $X \theta Y$ in $\text{closure}(AC(Q))$, where $\theta$ is $<$ or $\leq$, we introduce in $G(Q)$ two nodes labeled $X$ and $Y$, and an edge labeled $\theta$ from $X$ to $Y$. Clearly, if there is a path between two nodes $X_1$ and $X_2$ in $G(Q)$, we have $X_1 \leq X_2$.

Definition 2.1. (leq-set) [ALM06] Given a nondistinguished variable $X$ in a CQAC query $Q$, the leq-set (less-than-or-equal-to set) of $X$, includes all distinguished variables $Y$ of $Q$ that satisfy the following conditions: There exists a path from $Y$ to $X$ in the inequality graph $G(Q)$, and all edges on all paths from $Y$ to $X$ are labeled $\leq$. In addition, in all paths from $Y$ to $X$, there is no other distinguished variable except $Y$.

Definition 2.2. (<-set) Given a nondistinguished variable $X$ in a CQAC query $Q$, the < -set (less-than set) of $X$, includes all distinguished variables $Y$ of $Q$ that satisfy the following conditions: There exists a path from $Y$ to $X$ in the inequality graph $G(Q)$ (and all edges on all paths from $Y$ to $X$ are labeled either $\leq$ or $<$). In addition, in all paths from $Y$ to $X$, there is no other distinguished variable except $Y$.

The definitions of geq-set and >-set are analogous to the definitions of leq-set and <-set, respectively. Note that the >-set (<-set) of $X$ is a superset of the geq-set (leq-set) of $X$.

2.1.1 Containment of CQ queries

Definition 2.3. (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 $Q_1(D)$ to $Q_1$ on $D$ is a subset of the answer $Q_2(D)$ to $Q_2$ on $D$, that is, $Q_1(D) \subseteq Q_2(D)$.

The two queries $Q_1$ and $Q_2$ are said to be equivalent, denoted $Q_1 \equiv Q_2$, iff $Q_1 \sqsubseteq Q_2$ and $Q_2 \sqsubseteq Q_1$. 

$X < c$, $X \leq c$ (left semi-interval) or one of $X > c$, $X \geq c$ (right semi-interval).
Chandra and Merlin [CM77] have shown that a CQ query $Q_1$ is contained in another CQ query $Q_2$ of the same (head) arity if and only if there exists a containment mapping from $Q_2$ to $Q_1$. The containment mapping is a (body) homomorphism $h$ from the variables of $Q_2$ to the variables and constants of $Q_1$ and from the constants of $Q_2$ to themselves, such that for each subgoal $p(Z_1, \ldots, Z_n)$ of $Q_2$ it holds that $p(h(Z_1), \ldots, h(Z_n))$ is a subgoal of $Q_1$. In addition, for $h$ to be a containment mapping from $Q_2$ to $Q_1$, it must be that the list $(X_1, \ldots, X_k)$ of the variables and constants in the head of $Q_1$ be $(h(Y_1), \ldots, h(Y_k))$ (that is, $X_i = h(Y_i)$ for $\forall i \in \{1, \ldots, k\}$), where $Q_2(Y_1, \ldots, Y_k)$ is the head of $Q_2$.

**Theorem 2.1. (Containment-Mapping Theorem) [CM77]** Given CQ queries $Q_1$ and $Q_2$, $Q_1 \sqsubseteq Q_2$ iff there exists a containment mapping from $Q_2$ to $Q_1$.  

A containment mapping on symbols actually induces a mapping on subgoals, and vice-versa. Intuitively, when any source subgoal maps to a target subgoal, since both subgoals have the same predicate name, it means that the source subgoal is just a more generalized version of the target subgoal. For example, subgoal $p(X, Y, Z)$ is a generalized version of subgoal $p(X, Y, Y)$. In the context of contained rewritings, the more restrictive of the two, namely the target subgoal is said to cover the source subgoal as shown in Figure 2.1.

Figure 2.1: Mapping and covering of CQ query subgoals in the context of contained rewritings.

Given two CQ queries $Q_1$ and $Q_2$ (without ACs), [CM77] proved that checking whether $Q_1 \sqsubseteq Q_2$ is NP-complete. However, testing containment may not turn out to be too inefficient in cases such as when queries have a relatively small number of subgoals, or when there exist few pairs of subgoals with the same predicate name.
2.1.2 Containment of CQAC queries

The containment test for CQAC queries is more involved than the containment test for CQ queries. There are two ways to test the containment of CQAC query $Q_1$ in CQAC query $Q_2$ [GSUW94, Klu88]. We describe them very briefly (for more details see, e.g., [ALM04]).

The first test uses the notion of a canonical database: For each relational subgoal $p_i(X_i)$ 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 $X_i$ (i.e., in forming $t$ each variable in $X_i$ is “frozen” to a unique constant except that equated variables are frozen to the same constant and each constant in $X_i$ 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 $Q_1$. (In Definition 3.4 on page 22 we formalize the concept of a total order on a canonical database for a CQAC query.) The test says that $Q_1$ is contained in $Q_2$ if and only if $Q_2$ computes, on all the canonical databases of $Q_1$, all the head tuples of $Q_1$.

The second containment test, see Theorem 2.2, uses the notion of a normalized version of a CQAC query. An equivalent normalized version [Klu88, ALM03] (see beginning of Section 2.1) $Q'$ of a CQAC query $Q$ does not have constants or repetitions of variable names in relational subgoals and has compensating built-in equality conditions.

**Theorem 2.2.** [GSUW94, ALM04] For CQAC queries $Q_1$ and $Q_2$, $Q_1 \subseteq Q_2$ iff implication $I$ holds:

$$I : \beta'_1 \Rightarrow \mu_1(\beta'_2) \lor \ldots \lor \mu_k(\beta'_2)$$

where $\mu_i$’s are all the containment mappings from $Q'_2$ to $Q'_1$ and $\beta'_i$ is a conjunction of all the ACs in $Q'_i$, $i \in \{1, 2\}$. That is, the ACs in the normalized version $Q'_1$ of $Q_1$ logically imply (denoted “$\Rightarrow$”) the disjunction of the images of the ACs of the normalized version $Q'_2$ of $Q_2$ under each mapping $\mu_i$.

For CQ queries without ACs, the problem of checking containment is in NP [CM77] because we need to consider only all possible containment mappings. For CQAC queries however, the problem is in $\Pi^p_2$ [Ull00, vdM92], since we need to consider every possible containment mapping for every possible ordering of variables in the contained query.
2.2 Rewriting Queries Using Views

In this section, we briefly outline the problem of rewriting queries using views. We also introduce some key terms that have been used in existing work on this problem.

We consider the problem of finding rewritings under the closed-world assumption (CWA) [AD98], where for a given database, each view instance stores exactly the tuples satisfying the view definition. In addition, we consider contained rewritings under the open-world assumption (OWA) [AD98, LRO96]. Here, the views are sound but not necessarily complete, that is 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 restricted to using a set of views $V = \{V_1, \ldots, V_n\}$ defined on the schema of $D$. So instead of directly evaluating $Q$ on $D$, we rewrite $Q$ in terms of $V$ and then evaluate the rewriting on $D$. We consider the following types of rewritings $R$ of $Q$ using $V$. Here, $D_V$ is the result of adding to database $D$ the answers to views $V$ on $D$.

**Definition 2.4. (Rewritings)**

1. a. (CWA) $R$ is a contained rewriting of $Q$ using $V$ under the CWA if and only if $R(D_V) \subseteq Q(D)$ for all databases $D$.
   
b. (OWA) $R$ is a contained rewriting of $Q$ using $V$ under the OWA if and only if $R(I_V) \subseteq Q(D)$ for all databases $D$ and view instances $I_V$ such that $I_V \subseteq D_V$.

2. (CWA) $R$ is a containing rewriting of $Q$ using $V$ if and only if $Q(D) \subseteq R(D_V)$ for all databases $D$.

3. (CWA) $R$ is an equivalent rewriting of $Q$ using $V$ if and only if $Q(D) = R(D_V)$ for all databases $D$.

Since the answer to a containing rewriting $R$ on a database $D$ must contain all tuples that occur in the answer to $Q$ on $D$, 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 $Q$. At the same time, 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 $V$, for deciding whether there exists a contained (or containing) rewriting of $Q$ using $V$, we need to know the language in which we are allowed to construct rewritings. In the rest of this thesis we assume, unless otherwise stated, that the language of the rewritings is the language of UCQAC queries for contained rewritings and the language of CQAC queries for containing rewritings.

We define the expansion of a rewriting as follows:

**Definition 2.5. (Expansion of a rewriting)** Let $V$ be a set of CQAC views defined on a database schema $P$, and let $R$ be a CQAC rewriting defined on schema $V$. The expansion of $R$ is a CQAC query $R_{\text{exp}}$ that is obtained from $R$ as follows. For each subgoal $v(x_1,\ldots,x_r)$ of $R$, such that $v(x_1,\ldots,x_r)$ corresponds to view $V(y_1,\ldots,y_r) \in V$, replace $v(x_1,\ldots,x_r)$ in the body of $R$ 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 $R_{\text{exp}}$ is defined on the database schema $P$. □

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 2.3 is based on Definitions 2.4 and 2.5 and gives the tests for determining whether a CQAC rewriting $R$ is a contained (or containing) rewriting of a CQAC query $Q$ using CQAC views $V$.

**Theorem 2.3.** Let $Q,V_1,\ldots,V_n$ be CQAC queries defined on database schema $D$, and let $R$ be a CQAC rewriting of $Q$ using $\{V_1,\ldots,V_n\}$. Then

1. $R$ is a contained rewriting of $Q$ if and only if $R_{\text{exp}} \subseteq Q$.

2. $R$ is a containing rewriting of $Q$ if and only if $Q \sqsubseteq R_{\text{exp}}$. □
Contained rewritings are defined under the CWA as well as the OWA. But ascertaining whether or not a given rewriting \( R \) is contained in a given query \( Q \), involves only testing the containment of \( R^{exp} \) in \( Q \), as specified in Theorem 2.3. Irrespective of whether it is the CWA or the OWA, the expansion \( R^{exp} \) is the same. Hence, we have the following proposition.

**Proposition 2.1.** A CQAC rewriting of a CQAC query using CQAC views is a contained rewriting under the open-world assumption (OWA) iff it is a contained rewriting under the closed-world assumption (CWA). □

If CQAC queries \( R_1 \) and \( R_2 \) are two rewritings of a CQAC query \( Q \) using CQAC views \( V_1, \ldots, V_n \), and if \( R_1 \equiv R_2 \) (i.e., if we have equivalence when \( R_1, R_2 \) are expressed in terms of views \( V_1, \ldots, V_n \)), then \( R_1 \) and \( R_2 \) are said to be *equivalent as queries*. If, however, we have \( R_1^{exp} \equiv R_2^{exp} \) (i.e., if we have equivalence when \( R_1, R_2 \) are expressed in terms of base relations) then \( R_1 \) and \( R_2 \) are said to be *equivalent as expansions*. Whenever \( R_1 \) and \( R_2 \) are equivalent as queries, then they are also equivalent as expansions. But when \( R_1 \) and \( R_2 \) are equivalent as expansions, then it is not necessary that they are also equivalent as queries.
Chapter 3

Contained Rewritings

In this chapter, we describe our work on maximally contained rewritings. We present a sound and complete algorithm Build-MaxCR, for constructing a UCQAC size-limited maximally-contained rewriting (that is, an MCR that has up to a predetermined number of view literals) of CQAC queries using CQAC views. We start by providing a number of definitions (Section 3.1 on page 18 and Section 3.2 on page 22). This establishes the formal framework that we use to describe the subtleties of algorithm Build-MaxCR (Section 3.3 on page 26) and to prove its correctness (Section 3.7 on page 55). We also give illustrative examples (Section 3.4 on page 32), point out some salient features of Build-MaxCR (Section 3.5 on page 38), and discuss possible variations in the steps of Build-MaxCR (Section 3.6 on page 46). Chapter 5 (page 93) gives the results of our experiments on Build-MaxCR.

3.1 Maximally Contained Rewritings

This section defines maximally contained rewritings. It also gives the motivation for introducing the concept of $k$-boundedness in CQAC queries.

An algorithm trying to find a contained rewriting of a CQAC query $Q$ using a set of CQAC views $\mathcal{V}$, can try to minimize the false negatives that would be obtained by evaluating the rewriting, by constructing a rewriting that is maximally contained.

**Definition 3.1.** (Maximally contained rewriting) A query $R$ defined in query language $\mathcal{L}_1$ is a maximally contained rewriting (MCR) of a query $Q$ defined in language $\mathcal{L}_2$ using a
We can show that each of \( R \) is a contained rewriting of \( Q \) in terms of \( \mathcal{V} \), and (2) there exists no contained rewriting (in language \( \mathcal{L}_1 \)) \( R' \) of \( Q \) using \( \mathcal{V} \), such that the expansion of \( R \) is properly contained in the expansion of \( R' \). \( \square \)

Suppose we are given a CQAC query \( Q \) and a set \( \mathcal{V} \) of CQAC views, such that each of \( R_1 \) and \( R_2 \) is a CQAC contained rewriting of \( Q \) using \( \mathcal{V} \). It is easy to see that the union \( R_1 \cup R_2 \) is also a contained rewriting of \( Q \) using \( \mathcal{V} \). This observation motivates us to consider the language of unions of CQAC queries for maximally contained rewritings of CQAC queries using CQAC views. Thus, for the results in this chapter, each of \( \mathcal{L}_2 \) and \( \mathcal{L}_3 \) is the language of CQAC queries while \( \mathcal{L}_1 \) is the language of UCQAC queries.

The first question we examine is whether such a UCQAC MCR is always bounded in size. Consider an example based on the ideas from [ALM06].

**Example 3.1.** Let query \( Q \) and views \( V_1 \) and \( V_2 \) be defined as follows.

\[
Q() := p(X,Y), p(Y,Z), s(Y), X \geq 2, Z \leq 7.
\]

\[
V_1(L,M) := p(L,M), L \geq 2, M \leq 7.
\]

\[
V_2(A,C) := p(A,B), p(B,C), s(A), s(C).
\]

We can show that each of \( R_3 \) and \( R_4 \) (see expansions \( R_3^{exp} \) and \( R_4^{exp} \), respectively) is a CQAC contained rewriting of \( Q \) using \( V_1 \) and \( V_2 \):

\[
R_3() := V_1(L_1,A_1), V_2(A_1,C_1), V_1(C_1,M_2).
\]

\[
R_4() := V_1(X,T_1), V_2(T_1,T_2), V_2(T_2,T_3), V_1(T_3,Z).
\]

\[
R_3^{exp}() := p(L_1,A_1), p(A_1,B_1), s(A_1), p(B_1,C_1), p(C_1,M_2), s(C_1), L_1 \geq 2, A_1 \leq 7, C_1 \geq 2, M_2 \leq 7.
\]

\[
R_4^{exp}() := p(X,T_1), p(T_1,U_1), s(T_1), p(U_1,T_2), p(T_2,U_2), s(T_2), p(U_2,T_3), p(T_3,Z), s(T_3), X \geq 2, T_1 \leq 7, T_3 \geq 2, Z \leq 7.
\]

Further, one can use the template of \( R_3 \) and \( R_4 \) to build rewritings \( R_5 \) (that has one extra \( V_2 \) subgoal as compared to \( R_4 \)), \( R_6 \) (two extra \( V_2 \) subgoals), and so on.

\[
R_5() := V_1(X,T_1), V_2(T_1,T_2), V_2(T_2,T_3), V_2(T_3,T_4), V_1(T_4,Z).
\]

\[
R_6() := V_1(X,T_1), V_2(T_1,T_2), V_2(T_2,T_3), V_2(T_3,T_4), V_2(T_4,T_5), V_1(T_5,Z).
\]
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 the family of rewritings $\mathbf{R} = \{R_3, R_4, R_5, R_6, \ldots\}$ that we build in this manner, each rewriting $R_i$ (for $i \geq 3$) has two properties:

- the expansion of $R_i$ is contained in $Q$, and
- $R_i$ (for $i > 3$) is not contained in $R_j$ for any $3 \leq j < i$.

Therefore, a UCQAC maximally contained rewriting of $Q$ in terms of $\{V_1, V_2\}$ must include every $R_i$ in the infinite-cardinality family $\mathbf{R}$.

3.1.1 $k$-bounded Queries

The point of Example 3.1 (page 19) is that the number of CQAC disjuncts (such as $R_i$’s in that example) in the maximally-contained UCQAC rewriting of a CQAC query using CQAC views may not be bounded, provided that the language of rewritings is UCQAC. Hence an algorithm for finding the UCQAC MCR may not terminate on some CQAC inputs.

In this subsection, we address the problem by introducing the concept of size-limited MCRs. We then define the problem of constructing a UCQAC size-limited MCR for a CQAC query using CQAC views. We use the following definition:

**Definition 3.2. (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, the problem of constructing a UCQAC size-limited ($k$-bounded) MCR for a CQAC query using CQAC views is specified as follows:

1. 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.

2. The problem output is a UCQAC query $P = \bigcup_j P'_j$ in terms of $\mathcal{V}$, such that:
(a) $P^{exp}$ is contained in $Q$, $P^{exp} \subseteq Q$;
(b) $P$ is a $k$-bounded (UCQAC) query in terms of $\mathcal{V}$; and
(c) for each $k$-bounded UCQAC query $R$ in terms of $\mathcal{V}$ such that $R^{exp} \subseteq Q$, we have that $R^{exp} \subseteq P^{exp}$.

Let us revisit Example 3.1 (page 19) in the light of this problem definition. In that example, a 3-bounded MCR of $Q$ using $\mathcal{V}$ would contain just the CQAC $R_3$, a 5-bounded MCR of $Q$ using $\mathcal{V}$ would contain $R_3 \cup R_4 \cup R_5$, and so on.

The proposed algorithm Build-MaxCR solves the above problem for arbitrary inputs $(Q, \mathcal{V}, k)$ as defined in the problem formalization. Our soundness and completeness results for Build-MaxCR (Section 3.7 on page 55) 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.

In general, any MCR tries to approximate the answer to query $Q$ using the available views in $\mathcal{V}$, in cases where it may not be possible to obtain an exact equivalent rewriting of $Q$ using $\mathcal{V}$. Therefore just like the answer given by any other MCR, the answer given by the $k$-bounded MCR too, is an approximate answer to the query $Q$. By increasing $k$, the size-limited MCR can be made to approach closer and closer to $Q$, and if in fact $Q$ does have an equivalent rewriting of size $k$ or smaller, then algorithm Build-MaxCR is guaranteed to find it. Once an equivalent rewriting has been found, the algorithm terminates and does not continue to search for other MCRs. If no such equivalent rewriting exists, then the answer given by algorithm Build-MaxCR is the best approximation that can be constructed using rewritings of size up to $k$.

The tradeoff involved is that for a larger $k$, the algorithm Build-MaxCR that finds the $k$-bounded MCR needs to do more processing but may yield an answer that is closer to $Q$. The user can select $k$ in an online fashion, after seeing the CQAC MCRs as they are output one by one, and can choose to stop further iterations when the answer is reasonably close to $Q$, or when the answer tuples that have been generated by evaluating those rewritings are sufficiently large in number. Other criteria for determining the point of termination may include domain knowledge or previous usage statistics.
3.2 Preliminaries for Algorithm Build-MaxCR

In this section, we give the requisite definitions.

**Definition 3.3.** (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, we will need the notion of a canonical database $D_Q$ for a safe CQ query $Q$ (Section 2.1 on page 10). $D_Q$ 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 3.4.** (Canonical databases for a safe CQAC query; total order on a canonical database for a CQAC query) The set $D^Q$ 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 query $Q$. (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 $D_Q$. Specifically, canonical database $D_S$ results from applying the mapping $M$ associated with $S$ to the variables in $D_Q$. 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 $D_S$.)

\begin{definition}[Build-MaxCR problem input] The input for algorithm Build-MaxCR is the triple $(Q, \mathcal{V}, k)$, where $Q$ is a CQAC query, $\mathcal{V}$ is a set of CQAC views, and $k$ is a positive integer number.
\end{definition}

\begin{definition}[CQAC-rewriting template] For a given Build-MaxCR problem input $(Q, \mathcal{V}, k)$, a CQAC-rewriting template $P_i$ of size $t$ is a cross product of $t$ relational subgoals in terms of $\mathcal{V}$, such that $P_i = v_{i1}(\bar{x}_1) \times \ldots \times v_{it}(\bar{x}_t), t \leq k$ has a 1:1 association with multiset \{\{v_{i1}, \ldots, v_{it}\}\} of names of views in $\mathcal{V}$.
\end{definition}

In the definitions and results below, we use the following notation: For a Build-MaxCR problem input $(Q, \mathcal{V}, k)$, let $P$ be a CQAC-rewriting template, of some size $t \leq k$.

\begin{definition}[MaxCR canonical database for CQAC query $Q$ and CQAC-rewriting template $P$] The set $\mathcal{D}_Q^P$ 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 $\mathcal{D}^{(P_{exp})}$, see Definition 3.4 on page 22) is extended, for the construction of $\mathcal{D}_Q^P$, to include also all the numerical constants of the query $Q$.
\end{definition}

\begin{definition}[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 3.4 on page 22 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 $P^{exp}$.

**Definition 3.9. (Homomorphism property)** [ALM06] Let $Q_1$, $Q_2$ be two classes of CQAC queries. We say that containment testing on the pair $(Q_2, Q_1)$ has the homomorphism property if for any pair of queries $(Q_2, Q_1)$ with $Q_2 \subseteq Q_2$ and $Q_1 \subseteq Q_1$, the following holds: $Q_1 \sqsubseteq Q_2$ iff there is a homomorphism $\mu$ from $\text{core}(Q_2)$ to $\text{core}(Q_1)$ such that $AC(Q_1) \Rightarrow \mu(AC(Q_2))$, where the symbol $\Rightarrow$ denotes logical implication.

Although the homomorphism property is defined for two classes of queries, it is often referred to in the context of two queries, that is, the case in which each of the two classes contain exactly one query. For testing the containment of CQAC query $Q_1$ in CQAC query $Q_2$, 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 2.2 (page 14 in Section 2.1.2) 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 3.10. (Single-mapping contained rewriting)** A CQAC contained rewriting $R$ of a CQAC query $Q$ using a set of CQAC views $V$ is a single-mapping contained rewriting iff the homomorphism property holds for the containment of $R^{exp}$ in $Q$.

**Definition 3.11. (Multiple-mapping contained rewriting)** A CQAC contained rewriting $R$ of a CQAC query $Q$ using a set of CQAC views $V$ 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 3.2 below is a single-mapping MCR of $Q$ using $V$, while the $R$ in Example 3.3 is a multiple-mapping MCR.

---

1Recall from Section 2.1 (page 10) that $\text{core}(Q_2)$ denotes the relational part of $Q_2$. 

---
EXAMPLE 3.2.

\[ Q() \leftarrow p(A), \ A > 3. \]
\[ V() \leftarrow p(X), \ X > 3. \]
\[ R() \leftarrow V(). \]

\[ \Box \]

EXAMPLE 3.3.

\[ Q() \leftarrow p(A, B), \ A \leq B. \]
\[ V() \leftarrow p(X, Y), \ p(Y, X). \]
\[ R() \leftarrow V(). \]

\[ \Box \]

By the containment tests of \[\text{Ull00, vdM92}\] (see Section 2.1.2 on page 14) it holds that in Example 3.3, \( R \) is a contained rewriting of query \( Q \) using view \( V \). However, containment cannot be established using a single containment mapping \[\text{CM77}\] from \( Q \) to the expansion of \( R \).

The algorithm \( \text{Build-MaxCR} \) that we present is, to the best of our knowledge, the first algorithm to find the UCQAC maximally contained rewriting of CQAC queries and views for the general case which includes multiple-mapping contained rewritings.

3.3 Description of Algorithm \( \text{Build-MaxCR} \)

In this section, we provide a detailed description of the steps of algorithm \( \text{Build-MaxCR} \) and point out its subtleties. The working of \( \text{Build-MaxCR} \) is also summarized in the pseudocode in Algorithm 1 (page 28).

Algorithm \( \text{Build-MaxCR} \) accepts on input a CQAC query \( Q \), a set of CQAC views \( \mathcal{V} \), and a positive integer number \( k \). For an input \((Q, \mathcal{V}, k)\), the algorithm returns a UCQAC query \( P = \bigcup_j P'_j \) defined in terms of \( \mathcal{V} \), such that \( P^{\text{exp}} \) is contained in \( Q \), and such that \( P \) is a \( k \)-bounded (UCQAC) query in terms of \( \mathcal{V} \). Our soundness and completeness results for \( \text{Build-MaxCR} \), Theorem 3.1 (Section 3.7.2, page 60) and Theorem 3.2 (Section 3.7.3, page 62), establish that for each input \((Q, \mathcal{V}, k)\) described above, \( \text{Build-MaxCR} \) returns a maximally contained rewriting of \( Q \) in the language of \( k \)-bounded UCQAC queries over \( \mathcal{V} \), if such a rewriting exists. Example 3.4 (Section 3.4, page 32) and Example 3.5 (Section 3.4, page 34) provide specific illustrations of this general description of how the algorithm works.
The general idea of algorithm Build-MaxCR 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\footnote{In this thesis, 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.} such that $S'_j$ ensures containment of $\bar{P}_{\text{exp}} & S'_j$ 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 [Klu88], $\bar{P}_{\text{exp}} & S'_j \sqsubseteq Q$ for each $j$ ensures $P_{\text{exp}} \sqsubseteq Q$, where $P = \bigcup_j \bar{P}_j & S'_j$.)

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 $\mathcal{V}$. Each $P_i$ is a CQAC-rewriting template (for $Q$) of size $t$ (Definition 3.6 on page 24), with $t \leq k$. Note that each $P_i = \nu_{i_1}(\bar{x}_1) \times \ldots \times \nu_{i_t}(\bar{x}_t), t \leq k$, is associated with exactly one multiset $\{\{\nu_{i_1}, \ldots, \nu_{i_t}\}\}$ 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 $\{P_i = \nu_{i_1}(\bar{x}_1) \times \ldots \times \nu_{i_t}(\bar{x}_t), t \leq k\}$.

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 $P_{\text{exp}}^i$. This set of MaxCR canonical databases for $P_i$ and $Q$ (Definition 3.7, page 24) 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 Definition 3.4 on page 22 and Definition 3.7 on page 24) and for ensuring the positive outcome for the containment test for ($P'_j)^{\text{exp}} \subseteq Q$. The containment test, which is implicit in the algorithm (i.e., the outcome of the test is
Algorithm 1: The algorithm Build-MaxCR.

**Input**: CQAC query $Q$; set of CQAC views $\mathcal{V}$; $k \in \mathbb{N}$

**Output**: $k$-bounded UCQAC query $P = \bigcup P_j'$ in terms of $\mathcal{V}$, such that $P^\text{exp} \subseteq Q$

1. $P \leftarrow \emptyset$; // $P$ is a union of CQAC queries in the output of Build-MaxCR

2. for $t = 1$ to $k$ do

3. \hspace{1em} $\mathcal{P}_t \leftarrow$ set of all CQAC-rewriting templates of size $t$;

4. \hspace{1em} while $\mathcal{P}_t \neq \emptyset$ do

5. \hspace{2em} $P_t \leftarrow$ one template from $\mathcal{P}_t$; \hspace{1em} $\mathcal{P}_t \leftarrow \mathcal{P}_t - \{P_t\}$;

6. \hspace{2em} $\mathcal{D}_t \leftarrow$ the set of all MaxCR canonical databases for $P_t$ and $Q$ that make the head of $P_t^\text{exp}$ true;

7. \hspace{2em} $\mathcal{M} \leftarrow$ set of all mappings $\mu_{ij}$ from relational subgoals of $Q$ to same-name subgoals of $P_t^\text{exp}$;

8. \hspace{2em} if $\mathcal{M} = \emptyset$ then continue;

9. //Single-mapping processing:

10. \hspace{2em} for $j = 1$ to $|\mathcal{M}|$ do

11. \hspace{3em} produce $D_{ij} \subseteq D_i$, where each $D \in \mathcal{D}_t$ is included in $D_{ij}$ iff embedding $Q$ in $D$ using $\mu_{ij}$ makes the head of $Q$ true;

12. \hspace{3em} $S_{ij} \leftarrow$ set of summary ACs for $D_{ij}$; // $S_{ij}$ is the logical OR of all total orders for $D_{ij}$;

13. \hspace{3em} $\bar{x}_{ij} \leftarrow \tilde{\mu}_{ij}(\text{headVars}(Q));$ // $\mu_{ij}$ induces $\tilde{\mu}_{ij}$

14. \hspace{3em} if all variables used in $S_{ij}$ also occur in $P_t$ then

15. \hspace{4em} $P \leftarrow P \cup (P_t'(\bar{x}_{ij}) \triangleright P_t & S_{ij});$

16. //Multiple-mapping processing:

17. \hspace{2em} if for some set $J = \{j^{(1)}, \ldots, j^{(r)}, \ldots, j^{(m)}\}$ of the $j$’s above, such that $m > 1$, all the $\bar{x}_{ij(r)}$’s are the same, call them $\bar{x}_i$, then

18. \hspace{3em} $\hat{D}_i \leftarrow \bigcup_{r=1}^{m} D_{ij(r)}$; $\hat{S}_i \leftarrow$ set of summary ACs for $\hat{D}_i$;

19. \hspace{3em} if all variables used in $\hat{S}_i$ also occur in $P_t$ then

20. \hspace{4em} $P \leftarrow P \cup (P_t'(\bar{x}_i) \triangleright P_t & S_i);$  


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 $C_i$, 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 3.7 on page 24.) The basis for the definition of MaxCR canonical databases is the (standard) notion of the set $C'_i$ of canonical databases for $P_{i}^{\exp}$, see Definition 3.4 (page 22). By definition, each database $C' \in C'_i$ 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 $C_i$ of MaxCR canonical databases for $P_i$ and $Q$, we use exactly the same (i.e., the standard) database-construction process as for $C'_i$, except that we generate all the total orders based on the set $MaxCrVars = VarsConsts(P_{i}^{\exp}) \cup Consts(Q)$, where $Consts(Q)$ is the set of all constants in the definition of the input query $Q$. A total order associated with each $C \in C_i$ is the set of ACs on $MaxCrVars$ that yields database $C$ in the construction of $C_i$. As illustrated in Example 3.4 (page 32 in Section 3.4) and as shown in the proof of Theorem 3.2 (page 62 in Section 3.7.3), using the constants of $Q$ in the construction of $C_i$ ensures completeness of algorithm Build-MaxCR.

Algorithm Build-MaxCR works with a specific subset $D_i$ of all MaxCR canonical databases $C_i$ for $P_i$ and $Q$. Let us treat the CQAC-rewriting template $P_i$ as a CQ query without nondistinguished variables. Then $D_i$ is defined as the largest subset of $C_i$ such that each $D \in D_i$ 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 3.4 (page 32 in Section 3.4), 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 $C_i$, thus rendering $D_i$ a proper subset of $C_i$. (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 $D_i$ of MaxCR canonical databases for $P_i$ and $Q$ such that each $D \in D_i$ makes the head of $P_{i}^{\exp}$ true, the algorithm next determines which of the databases in $D_i$ 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 $D_{ij}$ of $D_i$ 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 $D_{ij}$ of MaxCR canonical databases for $P_i$ and $Q$. That is, $S_{ij}$ is a set of ACs that characterizes exactly the databases in $D_{ij}$, in the sense that $S_{ij}$ is the logical OR of the total-order ACs for the set $D_{ij}$. (Please see Example 3.5 on page 34 in Section 3.4 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 3.1 (page 60 in Section 3.7.2) we have that $(P'_{ij})^{\text{exp}}$ is contained in $Q$. (The head variables of $P'_{ij}$ are determined by Build-MaxCR in the process of constructing $D_{ij}$.) 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 $D_{ij}$ of MaxCR databases for $P_i$ and $Q$, such that each set $D_{ij}$ makes the heads of both $P_i^{\text{exp}}$ and $Q$ true. Recall that $D_i$ is the set of MaxCR canonical databases for $P_i$ and $Q$ such that each database in $D_i$ makes the head of $P_i^{\text{exp}}$ true. For each database $D \in D_i$, let us denote by $\iota$ the mapping that associates each relational subgoal $r$ of $P_i^{\text{exp}}$ with the stored tuple $t \in D$ such that $t$ is generated from $r$ by construction of database $D$. $\iota$ is the generative mapping from $P_i^{\text{exp}}$ to $D$. (See Definition 3.8 on page 24 in Section 3.2 and its discussion.) Note that $\iota$ induces a (consistent, in fact generative, see Definition 3.8 on page 24 in Section 3.2) assignment mapping $\tilde{\iota}$ from the variables of $P_i^{\text{exp}}$ to the stored values in $D$, with the property that $\tilde{\iota}(\text{headVars}(P_i^{\text{exp}}))$ is always an answer to $P_i^{\text{exp}}$ on the database $D$, by construction of the MaxCR canonical database $D$ for $P_i$ and $Q$. (Here, by $\text{headVars}(R)$ we denote the list of head variables of query $R$.) Recall that the head variables of $P_i^{\text{exp}}$ 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 $M$ be the set of all mappings $\mu_{ij}$ from the relational subgoals of query $Q$ to same-name subgoals of $P_i^{\text{exp}}$; algorithm Build-MaxCR associates one set of databases $D_{ij}$ with each $\mu_{ij} \in M$. (Example 3.4 on page 32 in Section 3.4 provides an illustration of mappings $\mu_{ij}$.) Fix an arbitrary mapping $\mu_{ij} \in M$ and an arbitrary database $D \in D_i$. 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 $P_i^{\text{exp}}$ with the generative mapping $\iota$ from $P_i^{\text{exp}}$ 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 \( \iota \) 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 \( D_{ij} \subseteq D_i \), 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 (page 28). For a fixed \( P_i \) and for a fixed mapping \( \mu_{ij} \in M \), Build-MaxCR considers all the databases in the set \( D_i \) of databases that make the head of \( P_i^{exp} \) true. Whenever, for a database \( D \in D_i \), 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 \( D_{ij} \) for \( P_i \) and \( \mu_{ij} \).

By construction, each database in the set \( D_{ij} \) makes the head of \( P_i^{exp} \) true, via the generative assignment mapping \( \iota \), and makes the head of \( Q \) true, via the assignment mapping \( \iota \circ \tilde{\mu}_{ij} \). Now Build-MaxCR uses each such \( D_{ij} \) to output, if possible, a single CQAC query \( P'_{ij} = P_i \& S_{ij} \), such that \( (P'_{ij})^{exp} \subseteq Q \) and such that \( S_{ij} \) is the logical OR of the total orders of exactly the databases in \( D_{ij} \). 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 3.1 (page 60 in Section 3.7.2), 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 \( (P'_{ij})^{exp} \subseteq Q \) is the above safety test for \( S_{ij} \) with respect to \( P'_{ij} \); 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 in the pseudocode of Algorithm 1 (page 28). Please see Example 3.5 (page 34 in Section 3.4) for an illustration. Essentially, even when some \( P'_{ij} \)'s 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\(^3\) of the sets \( J \) outlined in the pseudocode (line 17 in Algorithm 1 on page 28), by using bitmaps that encode whether the answer to the input query \( Q \) was empty or not “under” (in the sense discussed above) each mapping \( \mu_{ij} \) on each database \( D \) in the set \( D_i \) of databases that make the head of the query \( P_i^{exp} \) true.

3.4 Examples Illustrating Algorithm Build-MaxCR

Example 3.4 illustrates the flow of algorithm Build-MaxCR as described in Section 3.3 (page 26) and in Algorithm 1 (page 28). We use Example 3.5 (page 34) 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. This is followed by Example 3.6 (page 36) and Example 3.7 (page 36), which give additional examples of multiple-mapping processing. Finally, we include an example to show that MCRs for CQAC queries cannot be obtained by simply adding ACs to the MCRs of their CQ-counterparts. This highlights the need for an algorithm like Build-MaxCR.

Single-Mapping Example

EXAMPLE 3.4.

\[
Q(X) :: p(X), s(X), X < 3. \\
U(A) :: p(A), s(A). \\
V(N) :: p(N), N < 3. \\
W(L) :: s(L). \\
P'(A) :: U(A), A < 3 . \\
P''(N) :: V(N), W(N). \\
\]

In Example 3.4, suppose we are trying to find the \( k \)-bounded UCQAC MCR \( P \) of \( Q \) using \( V = \{U, V, W\} \) for \( k = 2 \). Initially, the outermost loop (line 2 in Algorithm 1 on page 28)
page 28) 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 $\text{MaxCrVars} = \text{VarsConsts}(P_i^{\exp}) \cup \text{Consts}(Q)$ is $\text{MaxCrVars} = \{A\} \cup \{3\} = \{A, 3\}$. Each possible total order on $\text{MaxCrVars}$ 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 Example 3.3 on page 34.) Also Build-MaxCR sets $\tilde{x}_{ij} = \tilde{\mu}_{ij}(\text{headVars}(Q)) = \tilde{\mu}(X) = A$. $S_{ij}$ 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 query $P_i^{ij}(\tilde{x}_{ij}) : = P_i \& S_{ij}$, that is the CQAC query $P'(A) : = U(A), A < 3$, is added to the union $P$.

Note that for this definition of $P'$, we have $(P')^{\exp} \subseteq Q$ while $(P_i)^{\exp} \nsubseteq Q$. 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 (Definition 3.7 on page 24), as opposed to “standard” canonical databases for the CQAC-rewriting templates. Also, note that in Example 3.4 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 $k$.

Now suppose Example 3.4 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, \(P_2\) is the CQAC-rewriting template \(P_t\), \(\mathcal{M} = \{\mu'\}\), where \(\mu' = \{p(X) \rightarrow p(N), s(X) \rightarrow s(L)\}\), and it induces \(\bar{\mu}' = \{X \rightarrow N, X \rightarrow L\}\). The set \(\text{MaxCrVars} = \text{VarsConsts}(P_i^{exp}) \bigcup \text{Consts}(Q)\) is \(\text{MaxCrVars} = \{N, L\} \bigcup \{3\} = \{N, L, 3\}\). 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, \(\mathcal{D}_1\) is set to \(\{D_{L<N<3}, D_{N<L<3}, D_{N<L<3}, D_{N<3=L}, D_{N<3<L}\}\). (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 \(\mathcal{D}_1\) 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 \(\mathcal{D}_1\) a proper subset of \(\mathcal{C}_i\).) Furthermore, out of the five canonical databases in \(\mathcal{D}_1\), there is only one canonical database, such that embedding \(Q\) in that database using \(\mu'\) makes the head of \(Q\) true. Hence the set \(\mathcal{D}_{ij}\) that 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 \(P_t\), the CQAC query \(P_i^{\prime}(\bar{x}_{ij}) :- P_t \& S_{ij}\), that is the CQAC query \(P_i^{\prime}(N) :- V(N), W(L), L = N, L < 3, N < 3\) with expansion \(P_i^{\prime \prime}(N) :- p(N), s(L), L = N, L < 3, N < 3\), 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 query \(P_i^{\prime}(N) :- V(N), W(N)\) that is added to the union \(P\).

**Multiple-Mapping Example**

**Example 3.5.**

\[Q() :- p(A, B), A \leq B.\]
\[V() :- p(X, Y), p(Y, X).\]
\[P() :- V().\]

\(\square\)

In Example 3.5, 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 (line 2 in Algorithm 1 on page 28), Build-MaxCR sets \( \mathcal{P}_1 = \{ V \} \). When \( V() \) is the CQAC-rewriting template \( \mathcal{P}_i \), the set \( \text{MaxCrVars} = \text{VarsConsts}(\mathcal{P}_i^{exp}) \cup \text{Consts}(\mathcal{Q}) \) is \( \text{MaxCrVars} = \{ X, Y \} \cup \emptyset = \{ X, Y \} \). Each possible total order on \( \text{MaxCrVars} \) yields exactly one canonical database in the set \( \mathcal{C}_i \) of MaxCR canonical databases for \( \mathcal{P}_i \) and \( \mathcal{Q} \). 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 \( \mathcal{P}_i^{exp} \) 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 \( \mathcal{Q} \) to the same-name subgoals of \( \mathcal{P}_i^{exp} \). In this case, \( \mathcal{M} = \{ \mu_1, \mu_2 \} \), where \( \mu_1 = \{ p(A, B) \rightarrow p(X, Y) \} \) and \( \mu_2 = \{ p(A, B) \rightarrow p(Y, X) \} \). \( \mu_1 \) and \( \mu_2 \) induce \( \tilde{\mu}_1 = \{ A \rightarrow X, B \rightarrow Y \} \) and \( \tilde{\mu}_2 = \{ A \rightarrow Y, B \rightarrow X \} \), 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 \( \mathcal{Q} \) in them using \( \mu_1 \) makes the head of \( \mathcal{Q} \) true, but for the remaining canonical database \( D_{X>Y} \) from \( \mathcal{D}_i \), embedding \( D_{X>Y} \) in \( \mathcal{Q} \) using \( \mu_1 \) does not make the head of \( \mathcal{Q} \) 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 \( \mathcal{S}_{ij} \) (of summary ACs that characterize exactly the canonical databases in \( \mathcal{D}_{ij} \)) as \( \mathcal{S}_{i1} = \{ X \leq Y \} \). Thus for the \( j = 1 \) case, this is an illustration of Build-MaxCR finding the set \( \mathcal{S}_{ij} \) 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 \( \mathcal{S}_{i2} = \{ X \geq Y \} \) 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 \( j = 2 \), \( \mathcal{S}_{ij} \) contains variables \( X \) and \( Y \), which do not occur in \( \mathcal{P}_i \). Thus \( \mathcal{P}_i \& \mathcal{S}_{i1} \) cannot be used to form a safe query \( \mathcal{P}'_{i1} \) that can be added to the union \( \mathcal{P} \). Similarly \( \mathcal{P}_i \& \mathcal{S}_{i2} \) too cannot be used to form a safe query \( \mathcal{P}'_{i2} \) that can be added to \( \mathcal{P} \). However, Build-MaxCR finds set \( J = \{ 1, 2 \} \), \( |J| > 1 \), for which \( \tilde{x}_{i1} = \tilde{\mu}_1(\text{headVars}(\mathcal{Q})) \) and \( \tilde{x}_{i2} = \tilde{\mu}_2(\text{headVars}(\mathcal{Q})) \) are the same. This is an illustration of the multiple-mapping processing done by Build-MaxCR (lines 16–20 in Algorithm 1 on page 28). For this set \( J \), Build-MaxCR finds \( \tilde{\mathcal{D}}_i = \mathcal{D}_{i1} \cup \mathcal{D}_{i2} = \{ D_{X<Y}, D_{X=Y} \} \) \( \cup \{ D_{X>Y}, D_{X=Y} \} = \{ D_{X<Y}, D_{X=Y}, D_{X>Y} \} \), and \( \tilde{\mathcal{S}}_i = \emptyset \) is the corresponding set of summary ACs for \( \tilde{\mathcal{D}}_i \). Now, for this \( \tilde{\mathcal{S}}_i \), it is true (trivially, since \( \tilde{\mathcal{S}}_i \) has no variables) that all variables used in \( \tilde{\mathcal{S}}_i \) also occur in \( \mathcal{P}_i = V() \). Hence Build-MaxCR adds the CQAC query
\[ P_i'() \leftarrow P_i \cap \bar{S}_i, \] 

that is, the CQAC query \( P'() \leftarrow V() \) to the union \( P \). This completes the Build-MaxCR processing for \( k = 1 \).

**Some Other Multiple-Mapping Examples of Applying Build-MaxCR**

Having presented a detailed description of multiple-mapping processing in Example 3.5 (page 34), here in addition we outline briefly, some simple illustrative examples of multiple-mapping contained rewritings. Note that these are obtained as answers by algorithm Build-MaxCR for even small values of \( k \) such as \( k = 1 \) but will be missed by any single-mapping algorithm. In both examples, Example 3.6 and Example 3.7, \( R \) is a multiple-mapping contained rewriting of \( Q \) using \( V \) that cannot be obtained by any single-mapping algorithm.

**EXAMPLE 3.6.**

\[
Q() \leftarrow p(X_1, X_2), p(X_2, X_3), p(X_3, X_1), X_1 < X_2.
\]

\[
V() \leftarrow p(Y_1, Y_2), p(Y_2, Y_3), p(Y_3, Y_1), Y_1 < Y_3.
\]

\[
R() \leftarrow V().
\]

**EXAMPLE 3.7.**

\[
Q() \leftarrow p(A, B, 9), A < 9, B > 9.
\]

\[
V() \leftarrow p(E, X, 9), p(F, G, X), p(X, H, 9), E < 9, F < 8, G > 10, H > 9.
\]

\[
R() \leftarrow V().
\]

**Non-Alignment of CRs and MCRs**

We conclude this section on examples of Build-MaxCR by presenting an example to illustrate what we call the *non-alignment of CRs and MCRs*. The example shows that contained rewritings (CRs) for CQAC queries cannot be obtained by simply adding ACs to the (significantly more easy to find) contained rewritings of their CQ-counterparts. This emphasizes the need for an algorithm like Build-MaxCR.

In Example 3.8 let \( Q \) be a CQAC query that is to be answered using \( V = \{V_1, V_2\} \), where \( V_1 \) and \( V_2 \) are CQAC views. Consider rewritings \( R_1 \) and \( R_2 \) of \( Q \) using \( V \).

**EXAMPLE 3.8.**

\[
Q() \leftarrow p(X, Y), X < Y.
\]

\[
V_1() \leftarrow p(X, Y), X > Y.
\]
\[ V_2() \vdash p(X,Y), r(Y), X < Y. \]
\[ R_1() \vdash V_1(). \]
\[ R_2() \vdash V_2(). \]
\[ R_{1 \text{exp}}() \vdash p(X,Y), X > Y. \]
\[ R_{2 \text{exp}}() \vdash p(X,Y), r(Y), X < Y. \]

\( R_1 \) is not a contained rewriting of \( Q \) (since the AC part of \( R_{1 \text{exp}} \) is \( X > Y \) and the AC part of \( Q \) is \( X < Y \)). But rewriting \( R_2 \) is a contained rewriting of \( Q \). Since there is no other CR, say \( R_3 \), of \( Q \) using \( V \), such that \( R_3 \) contains \( R_2 \), the CR \( R_2 \) is, in fact, an MCR of \( Q \) using \( V \). \( Q_0 \) is the core of \( Q \). Let \( V_0 = \{ V_{10}, V_{20} \} \) be the set of views obtained by considering only the cores of the views in \( V \). In general, while forming \( V_0 \), for each view, the exportable (Section 2.1 on page 10) variables in that view are put into the view head. Consider rewritings \( R_{10} \) and \( R_{20} \) of \( Q_0 \) using \( V_0 \).

\[ Q_0() \vdash p(X,Y). \]
\[ V_{10}() \vdash p(X,Y). \]
\[ V_{20}() \vdash p(X,Y), r(Y). \]
\[ R_{10}() \vdash V_{10}(). \]
\[ R_{20}() \vdash V_{20}(). \]
\[ R_{10 \text{exp}}() \vdash p(X,Y). \]
\[ R_{20 \text{exp}}() \vdash p(X,Y), r(Y). \]

Both \( R_{10} \) and \( R_{20} \) are CRs of \( Q_0 \) using \( V_0 \). However, \( R_{20} \) is itself contained in \( R_{10} \). Thus, \( R_{20} \) is a CR of \( Q_0 \) but not an MCR of \( Q_0 \). \( R_{10} \) is a CR of \( Q_0 \) and is also an MCR of \( Q_0 \). Example 3.8 shows the non-alignment of CRs and MCRs because:

- \( R_2 \) is an MCR of \( Q \) using \( V \) but \( R_{20} \) is not an MCR of \( Q_0 \) using \( V_0 \).
- \( R_1 \) is not an MCR of \( Q \) using \( V \) but still \( R_{10} \) is an MCR of \( Q_0 \) using \( V_0 \).

In other words, we have the example of \( R_{20} \) that is a CR of \( Q_0 \) (although it is not an MCR of \( Q_0 \)), but when considered together with its AC part, \( R_2 \) is indeed an MCR of \( Q \). On the other hand, we also have the example of \( R_{10} \) that is indeed an MCR of \( Q_0 \), but when considered together with its AC part, \( R_1 \) is not an MCR of \( Q \).
3.5 Salient Features of Algorithm Build-MaxCR

The detailed discussion in Section 3.3 (page 26) points out several subtleties of Build-MaxCR. In addition to these, the processing steps employed by Build-MaxCR involve some special features that are required in order to preserve correctness and to help improve efficiency. In this section, we present some such features.

Reducing the Number of Canonical Databases that Need to be Processed by Build-MaxCR

Recall from Section 3.3 (page 26) that algorithm Build-MaxCR generates the set \( C_i \) of MaxCR canonical databases for \( P_i \) and \( Q \) by considering all the total orders on the set \( \text{MaxCrVars} = \text{VarsConsts}(P_i^{\text{exp}}) \cup \text{Consts}(Q) \), where \( \text{VarsConsts}(P_i^{\text{exp}}) \) is the set of variables and constants in the body of \( P_i^{\text{exp}} \) and \( \text{Consts}(Q) \) is the set of all constants in the definition of the input query \( Q \). It turns out that in place of \( \text{MaxCrVars} \) if Build-MaxCR uses the subset \( \text{MaxCrVars}' \) of \( \text{MaxCrVars} \), the algorithm still remains correct. Here, \( \text{MaxCrVars}' = \text{SelectedVarsConsts}(P_i^{\text{exp}}) \cup \text{Consts}(Q) \), and the set \( \text{SelectedVarsConsts}(P_i^{\text{exp}}) \) includes exactly the following:

1. All the constants in \( P_i^{\text{exp}} \).
2. All the variables in \( P_i \).
3. All the variables that occur two or more times in \( P_i^{\text{exp}} \). (This includes variables that occur two or more times in the relational subgoals in \( P_i^{\text{exp}} \) as well as those variables that occur exactly one time in some relational subgoal in \( P_i^{\text{exp}} \) but also occur one or more times in the arithmetic subgoals in \( P_i^{\text{exp}} \).)
4. All the variables that occur in the \(<\)-set\(^4\) as well as all the variables that occur in the \(>\)-set of any variable that is included in item 3.

In the worst case, the subset \( \text{MaxCrVars}' \) of \( \text{MaxCrVars} \) may be identical to \( \text{MaxCrVars} \), but in practice it is often smaller than the set \( \text{MaxCrVars} \). For \( n \) variables and constants, the number of total orders is \( O(n!) \). Hence using \( \text{MaxCrVars}' \) in place of

---

\(^4\)For definitions of \(<\)-set (Definition 2.2), \(\leq\)-set (Definition 2.1), \(>\)-set, and \(\geq\)-set, please see page 12 in Section 2.2. [ALM06] has more details.
MaxCrVars may significantly decrease the number of canonical databases that algorithm Build-MaxCR has to process, and may thus allow the algorithm to run more efficiently. In going from MaxCrVars to MaxCrVars', we are basically dropping all non-shared variables that are neither distinguished nor distinguishable. (Among the non-shared variables, the distinguished ones are included by item 2 and the distinguishable ones are included by item 4.) Intuitively, each canonical database represents one “case” and the variables that we drop in going from MaxCrVars to MaxCrVars' are precisely those that do not help in distinguishing some “case” from some other “case”. Hence they are not useful to the algorithm and can be dropped.

Finding the Appropriate Mappings that are to be Considered Collectively in Multiple-Mapping Processing

For its multiple-mapping processing, algorithm Build-MaxCR constructs the set $J$ (line 17 of Algorithm 1 on page 28) that allows it to form summary ACs $\tilde{S}_i$. Multiple-mapping processing in Build-MaxCR gives it additional chances to possibly have summary ACs whose variables come exclusively from the variables in $P_i$. Having such summary ACs is useful because these ACs can be conjoined with $P_i$ to form a valid (safe) CQAC query $P'_i$, which can then be added to $P$ (line 20 of Algorithm 1 on page 28) to form the UCQAC MCR of $Q$ using $V$. In this subsection, we discuss some strategies that are useful in finding set $J$.

In order to find any $J$, the first step is to partition $\mathcal{M}$ into groups of mappings. Groups are formed by the following rule. Mappings from $\mathcal{M}$ that produce identical $\bar{x}_{ij}$’s are placed in the same group, and mappings that produce different $\bar{x}_{ij}$’s are placed in different groups. Consider Example 3.9 which illustrates the formation of groups.

EXAMPLE 3.9.

\begin{align*}
Q(X) & \triangleright p(X), X < 3. \\
V(A, B) & \triangleright s(A, B), p(A), p(B). \\
R_1(A_1) & \triangleright V(A_1, B_1), A_1 < 3. \\
R_2(B_2) & \triangleright V(A_2, B_2), B_2 < 3. \\
R & = R_1 \cup R_2.
\end{align*}

In Example 3.9, $R$ is a UCQAC MCR of $Q$ using $V$. The mapping $p(X) \rightarrow p(A)$
Table 3.1: Finding summary ACs for Build-MaxCR.

<table>
<thead>
<tr>
<th>If $B_{\mu}$ represents ACs of type</th>
<th>Then include the following in summary ACs $S_{\mu}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X &lt; Y$, $X = Y$, $X &gt; Y$</td>
<td>Add nothing</td>
</tr>
<tr>
<td>$X &lt; Y$, $X = Y$</td>
<td>$X \leq Y$</td>
</tr>
<tr>
<td>$X &gt; Y$, $X = Y$</td>
<td>$X \geq Y$</td>
</tr>
<tr>
<td>$X &lt; Y$, $X &gt; Y$</td>
<td>$X \neq Y$</td>
</tr>
<tr>
<td>$X &lt; Y$</td>
<td>$X &lt; Y$</td>
</tr>
<tr>
<td>$X &gt; Y$</td>
<td>$X &gt; Y$</td>
</tr>
<tr>
<td>$X = Y$</td>
<td>$X = Y$</td>
</tr>
</tbody>
</table>

from $\mathcal{M}$ is placed in one group and produces the CQAC MCR $R_1$, which is added to the union $R$. The mapping $p(X) \rightarrow p(B)$ is placed in the other group and produces the CQAC MCR $R_2$, which is also added to $R$.

Consider bitmap $B$ that has exactly one bit for each MaxCR canonical database in $D_i$. For any mapping $\mu$ in $\mathcal{M}$ we have a corresponding bitmap $B_{\mu}$ obtained from $B$ as follows. For any MaxCR canonical database $D$ that is represented in $B$, the bit representing $D$ in $B_{\mu}$ is set to 1 if and only if embedding $Q$ in $D$ using $\mu$ makes the head of $Q$ true. For any bitmap $B_{\mu}$, $S_{\mu}$ is the equivalent representation of $B_{\mu}$ in the form of summary ACs and it is obtained by using the rules of logical ORing of ACs outlined in Table 3.1. Let $X$ and $Y$ be symbols (variables/constants) in $MaxCrVars$. If the total order of any MaxCR canonical database represented by a 1 in $B_{\mu}$ implies the AC $X < Y$ (or $X > Y$ or $X = Y$, respectively), then we say that $B_{\mu}$ “represents” ACs of type $X < Y$ (or of type $X > Y$ or type $X = Y$, respectively). For every pair of variables $X$ and $Y$ in $MaxCrVars$, examine $B_{\mu}$ to find all the types of ACs that are represented by $B_{\mu}$, and use them to locate the appropriate row in Table 3.1. Then from the right-hand column of the same row, obtain the summary ACs (if any) on $X$ and $Y$, that are to be included in $S_{\mu}$.

The following processing is applied to each group $G$. For each mapping $\mu$ in $G$, we have now obtained a corresponding $B_{\mu}$ and a corresponding $S_{\mu}$. Assume that initially all the $S_{\mu}$’s corresponding to all the $\mu$’s in that $G$ form one large fragment. For every pair of symbols $X$ and $Y$ in $MaxCRVars$, split each fragment into two sub-fragments — one sub-fragment consisting of all those $S_{\mu}$’s that include an AC relating $X$ and $Y$, and the

---

5Since each MaxCR canonical database has an associated total order, for that MaxCR canonical database exactly one of $X < Y$, $X > Y$, or $X = Y$ has to be true.
other sub-fragment consisting of all the remaining $S_\mu$’s from the fragment that was being split. After the splitting caused by all symbol pairs is done, the resulting fragments that are obtained are considered for processing first one-at-a-time, then two-at-a-time, and so on, until they are finally considered all-at-a-time. For any such collection of fragments that is being considered for processing, do the following. For every pair of symbols $X$ and $Y$ that are related by an AC in any $S_\mu$ in any fragment in that collection, logically OR together all the ACs on $X$ and $Y$ that occur in any $S_\mu$ in any fragment in that collection. Table 3.1 can be used to obtain the AC (if any) that results from this ORing. Let $X \text{ op } Y$ represent the resulting AC. A conjunction of the $X \text{ op } Y$ ACs obtained from all symbol pairs such as the pair $X$ and $Y$, gives the resulting summary ACs $\hat{S}_i$ (line 18 of Algorithm [1] on page 28). If all variables that occur in $\hat{S}_i$ also occur in $P_i$ (line 19 of Algorithm [1] on page 28), then all the fragments in the collection of fragments that was currently being processed (i.e., the collection of fragments that gave us the summary ACs $\hat{S}_i$) are removed from the pool of available fragments. That way, these same fragments do not have to be considered ever again, at the time when other larger-sized collections of fragments are being considered. (Recall that collections of fragments are considered in ascending order of their size, that is, collections of size one, collections of size two, and so on.)

Examples [3.10] and [3.11] illustrate the fragmentation procedure.

**EXAMPLE 3.10.**

\[ Q() \leftarrow p(W, X, Y, Z), W < 5. \]
\[ V(A, D) \leftarrow p(A, B, B, C), p(D, E, F, F). \]
\[ R = R_1 \cup R_2. \]
\[ R_1() \leftarrow V(A, D), A < 5. \]
\[ R_2() \leftarrow V(A, D), D < 5. \]
\[ R_3() \leftarrow V(A, D), A < 5, D < 5. \]

In Example [3.10], $R$ is the UCQAC MCR of $Q$ using $V$. The two mappings $p(W, X, Y, Z) \rightarrow p(A, B, B, C)$ and $p(W, X, Y, Z) \rightarrow p(D, E, F, F)$ (that are in the same group) have their corresponding summary ACs $S_\mu$ as $A < 5$ and $D < 5$, respectively. Note that $A < 5$ and $D < 5$ belong to different fragments. Thus the fragmentation procedure allows us to obtain the individual CQAC MCRs $R_1$ and $R_2$ that are present in $R$. Although $R_3$ is a contained rewriting of $Q$ using $V$, it is not a maximally contained rewriting, that is, $R_3$ is not a CQAC
MCR of $Q$ using $V$. This is because $R_3$ is contained in each of $R_1$ and $R_2$.

**EXAMPLE 3.11.**

\[
Q() \quad :\quad p(A, B), p(B, C), p(C, A), A < B.
\]

\[
V(L, M, N) \quad :\quad p(L, M), p(M, N), p(N, L).
\]

$R = R_1 \cup R_2 \cup R_3$.

\[
R_1() \quad :\quad V(L, M, N), L < M.
\]

\[
R_2() \quad :\quad V(L, M, N), M < N.
\]

\[
R_3() \quad :\quad V(L, M, N), N < L.
\]

In Example 3.11 at $k = 1$, Build-MaxCR obtains $R$ as the UCQAC MCR of $Q$ using $V$. The rewriting $R = R_1 \cup R_2 \cup R_3$ means that for the answer to $R$ to be true, MaxCR canonical databases with any total ordering on $L, M$, and $N$ are acceptable, except for the one MaxCR canonical databases with total ordering $L = M = N$. Note that obtaining $R$ was possible due to the formation of three different fragments, each giving us one of $R_1, R_2$, and $R_3$.

**Handling Head Variables that are not in the CQAC-Rewriting Template**

If the head variables $\bar{x}_{ij}$ (line 13 in Algorithm 1 on page 28) contain one or more variables that do not occur in the CQAC-rewriting template $P_i$ (although all variables in $\bar{x}_{ij}$ always occur in $P_i^{exp}$), then Build-MaxCR takes the following steps in an attempt to transform the $\bar{x}_{ij}$ into $\bar{x}'_{ij}$ such that all variables in $\bar{x}'_{ij}$ occur in $P_i$. Thus it may still be possible for Build-MaxCR to use the $P_i$ that is being considered to form a valid (safe) CQAC query that can be added to the UCQAC MCR $P$.

Let $\mu$ be a mapping such that $\mu(\text{headVars}(Q))$ gives us $\bar{x}_{ij}$. In single-mapping processing there will be just one such mapping, while in multiple-mapping processing all such mappings will belong to the same group of mappings. Let $B_{\bar{x}_{ij}}$ be the result of OR-ing together all the bitmaps corresponding to all such mappings $\mu$. For every variable $X$ in $\bar{x}_{ij}$ that does not occur in $P_i$, and for every variable $Y$ that does occur in $P_i$ and for which $X = Y$ is true (in the total order of at least one of the MaxCR canonical databases represented by a 1) in bitmap $B_{\bar{x}_{ij}}$, Build-MaxCR tries to replace $X$ by $Y$ throughout, and thus obtains $\bar{x}'_{ij}$ in place of $\bar{x}_{ij}$. Build-MaxCR also considers $B_{\bar{x}'_{ij}}$ in place of $B_{\bar{x}_{ij}}$, where the MaxCR canonical databases represented in $B_{\bar{x}'_{ij}}$ are a subset of those in $B_{\bar{x}_{ij}}$. Specif-
ically, \( B_{\bar{x}_{ij}} \) represents only those MaxCR canonical databases that are in \( B_{\bar{x}_{ij}} \) and whose total order implies \( X = Y \). Thus, Build-MaxCR tries to obtain a CQAC query whose head variables \( \bar{x}_{ij}' \) no longer contain variable \( X \) and whose summary ACs are a representation of \( B_{\bar{x}_{ij}} \) (and not \( B_{\bar{x}_{ij}} \)).

**Handling Variables from Summary ACs that are not in the CQAC-Rewriting Template**

When summary ACs, say \( \tilde{S}_i \), have been determined, some variables that occur in \( \tilde{S}_i \) may not be present in the CQAC-rewriting template \( P_i \) (although they are always present in \( P_i^{exp} \)). In such cases, conjoining the ACs in \( \tilde{S}_i \) with \( P_i \) cannot form a valid (safe) CQAC query, and the summary ACs \( \tilde{S}_i \) are discarded, since they are not useful in producing a rewriting. The only exception to this is when it is possible to transform \( \tilde{S}_i \) into some valid \( \tilde{S}_i' \), as explained next.

For every variable \( X \) in \( \tilde{S}_i \) that is not a head variable of any of the viewheads in \( P_i \), and for every AC \( X \ op \ Y \) in \( \tilde{S}_i \) in which \( X \) appears, Build-MaxCR takes the following steps, in an attempt to transform \( \tilde{S}_i \) into a valid set of ACs \( \tilde{S}_i' \) that can be conjoined with \( P_i \) to obtain a valid CQAC query. Note that here \( op \) is one of \{<, \leq, >, \geq, =, \neq \}, and \( Y \) is either a variable or a constant.

If \( op \) is < or \( \leq \): For every variable \( Z \) in either the geq-set [ALM06] or the >-set (page 12 in Section 2.1) of \( X \), the algorithm tries to transform \( \tilde{S}_i \) by replacing the AC \( X \ op \ Y \) by the AC \( Z \ op \ Y \).

If \( op \) is > or \( \geq \): For every variable \( Z \) in either the leq-set [ALM06] or the < -set (page 12 in Section 2.1) of \( X \), the algorithm tries to transform \( \tilde{S}_i \) by replacing the AC \( X \ op \ Y \) by the AC \( Z \ op \ Y \).

If \( op \) is = : For every variable \( U \) in the leq-set of \( X \) and for every variable \( W \) in the geq-set of \( X \), the algorithm tries to transform \( \tilde{S}_i \) by replacing the AC \( X \ op \ Y \) by the ACs \( U = W \) and \( U = Y \) (or \( W = Y \)).

If \( op \) is \( \neq \): For every variable \( U \) in the leq-set of \( X \) and for every variable \( W \) in the geq-set
of $X$, the algorithm tries to transform $\bar{S}_i$ by replacing the AC $X \text{ op } Y$ by the ACs $U = W$ and $U \neq Y$ (or $W \neq Y$).

The algorithm Build-MaxCR tries all possible ways of transforming $\bar{S}_i$ according to the rules above. Every transformed set $\bar{S}'_i$, in which the ACs in $\bar{S}'_i$ are consistent with each other and with the ACs in $P^{exp}_i$, is a valid $\bar{S}'_i$ that can be conjoined with $P_i$ in an attempt to form a valid CQAC query, and Build-MaxCR tries all such cases.

**EXAMPLE 3.12.**

\[
\begin{align*}
Q() & : p(X,Y), \ X < 5. \\
V(B) & : p(A,B), \ A \leq B. \\
R() & : V(B), \ B < 5. 
\end{align*}
\]

In Example 3.12, the set of summary ACs initially discovered by Build-MaxCR contains exactly one AC — the AC $A < 5$. In this case, the CQAC-rewriting template $P_i$ that is being considered is $V$. If $A$ was a head variable in view $V$, then $A$ would have been available in $P_i$, and we would have been able to conjoin the AC $A < 5$ with $P_i$ to get a valid CQAC query that is contained in $Q$. However, in this case $A$ is not available in the head of $V$ and hence we cannot access it directly. But observe that $V$ does contain an AC $A \leq B$, and that $B$ is available in the head of $V$ (i.e., $B$ is a variable in the geq-set of $A$). So even though we cannot add the AC $A < 5$, we can still add the AC $B < 5$, and this AC $B < 5$ together with the AC $A \leq B$, which is already in the body of $V$, produces the same effect as using the single AC $A < 5$ to constrain the values that can be taken by variable $A$. Hence Build-MaxCR transforms its initial set of summary ACs so that the AC $A < 5$, which was in its initial set, is now replaced by the AC $B < 5$. Thus, with the transformed summary ACs, Build-MaxCR is able to correctly find $R$ as a contained rewriting of $Q$ using $V$.

**Adding CQAC Contained Rewritings to the Final UCQAC Answer of Build-MaxCR**

When a valid contained rewriting of $Q$, say $P'$, is discovered, Build-MaxCR adds the CQAC $P'$ to its UCQAC answer $P$ as shown in line 15 and in line 20 of Algorithm 1 (page 28). In the process of adding $P'$ to $P$, Build-MaxCR takes the steps outlined in
Algorithm 2: Subroutine to add CQAC rewritings to P

**Input**: UCQAC query $P_{\text{input}}$, CQAC query $P'$, CQAC query $Q$

**Output**: UCQAC query $P_{\text{output}}$

begin

1. if $Q \sqsubseteq P'$ then
   
   2. Set $P_{\text{output}}$ to be the single CQAC query $P'$;
   3. Output $P_{\text{output}}$ as the final answer of Build-MaxCR and stop;

4. $n \leftarrow$ number of CQAC rewritings in $P_{\text{input}}$, where UCQAC query $P_{\text{input}} = P'_1 \cup P'_2 \cup \ldots \cup P'_n$ for CQAC queries $P'_1, P'_2, \ldots, P'_n$;

5. for $i = 1$ to $n$ do
   
   6. if $P' \sqsubseteq P'_i$ then
      
      7. $P_{\text{output}} \leftarrow P_{\text{input}}$;
      8. Return $P_{\text{output}}$;

7. for $i = 1$ to $n$ do
   
   8. if $P'_i \sqsubseteq P'$ then
      
      9. Remove $P'_i$ from $P_{\text{input}}$;
      10. $P_{\text{output}} \leftarrow P_{\text{input}} \cup P'$;

11. Return $P_{\text{output}}$;

end
Algorithm 2. $P_{input}$ denotes the UCQAC query $P$ that is input to Algorithm 2, and $P_{output}$ denotes the updated UCQAC query $P$ that is output by Algorithm 2. As indicated in this subroutine (Algorithm 2) of Build-MaxCR, if $Q \sqsubseteq P'$, it means that $P' \equiv Q$. Hence Build-MaxCR sets $P_{output}$ to be just the single CQAC query $P'$, returns $P_{output}$ as its final answer, and stops all further processing. Otherwise, if the existing UCQAC query $P_{input}$ is such that $P_{input} = P'_1 \cup P'_2 \cup \ldots \cup P'_n$ for CQAC queries $P'_1, P'_2, \ldots, P'_n$, then Build-MaxCR checks if $P'$ is contained in any of $P'_1, P'_2, \ldots, P'_n$. If it is contained, then Build-MaxCR does not add $P'$ at all, and returns $P_{input}$ itself as $P_{output}$. Otherwise, it proceeds to check each CQAC query in $P_{input}$ to see if it is contained in $P'$. Every CQAC query $P'_i$ in $P_{input}$, which has $P'_i \subseteq P'$ as true, is removed from the union $P_{input}$, following which $P'$ is added to the union $P_{input}$ to form $P_{output}$.

Thus, Algorithm 2 has three parts. In the first part, if the subroutine finds that $P'$ is equivalent to $Q$, it concludes that there is no need to search any further (because no other rewriting can be more “maximally” contained in $Q$). In the second part, if it finds that all cases handled by $P'$ are already being handled by some other single CQAC query that is in $P_{input}$, it does not add $P'$ to the union since doing so would be redundant. In the third part, if the subroutine finds any CQAC query $P'_i$ in the UCQAC query $P_{input}$, such that $P'_i$ is contained in $P'$, then it realizes that all cases handled by $P'_i$ are also handled by $P'$. Since it would be adding $P'$ to the union, it decides to delete the $P'_i$, which is going to be rendered redundant by the addition of $P'$.

### 3.6 Possible Variations on the Steps of Build-MaxCR

This section includes a discussion on some design decisions for Build-MaxCR, such as the choice of the direction for its outermost loop. It includes suggestions on the use of a separate module for finding single-mapping MCRs in parallel with a Build-MaxCR module, and discusses a preprocessing step that can potentially reduce the size of the input viewset. Finally, it outlines a brute-force counterpart for algorithm Build-MaxCR, which is helpful in appraising the efficiency gains that are obtained by the use of Build-MaxCR.

---

6It checks containment as queries (and not containment as expansions) assuming the open-world case. Under the CWA, it would have to compute (or store) the expansion of each CQAC query in $P$ and would then have to check for containment as expansions.
Varying the Direction for the Outermost Loop of Build-MaxCR

The outermost loop of algorithm Build-MaxCR (the \( t \)-loop starting in line 2 of Algorithm 1 on page 28) goes from 1 to \( k \), rather than going in the other direction from \( k \) to 1. We outline some reasons for this choice.

On many inputs, it is possible that at a low value of \( t \) itself, we end up getting a rewriting that is equivalent to \( Q \). Once this is found, Build-MaxCR stops all further processing. Hence, in cases where an equivalent rewriting can be found at a lower value of \( t \), if \( k \) is set to a high value, then the algorithm may end up doing a lot of extra work if the \( t \)-loop runs in the opposite direction. This is especially true at higher values of \( t \), where typically, the size of set \( MaxCRVars \) (Section 3.3, page 29) would be much larger than the size at lower values of \( t \). A large size of set \( MaxCRVars \) means that there will be a large number of MaxCR canonical databases that will need to be processed. In addition, the number of CQAC-rewriting templates as well as the size of each individual template increases with an increase in \( t \), and this too causes the algorithm to require more processing time.

The other advantage of starting at \( t = 1 \) and then increasing the value by 1 at each step is that the user does not have to make an a priori decision about the value of \( k \). The user can select \( k \) in an online fashion or can initially set \( k \) to a very high value and then choose to terminate the algorithm early after seeing the rewritings that Build-MaxCR adds to \( P \) for the lower values of \( t \), or after examining the results that are obtained by running those rewritings. Thus, users can employ their own criteria for deciding whether, for example, the answer is close enough to what they require, or whether the answer tuples that have been generated on running the (CQAC) rewritings obtained at lower values of \( t \) are sufficiently large in number. As a generalization, instead of running from 1 to \( k \), the \( t \)-loop of Build-MaxCR can in fact be made to run from any lower bound \( k_1 \) to any upper bound \( k_2 \). Moreover, instead of always giving a range, Build-MaxCR can even be made to process specific point values for \( t \) (for example, by having line 2 in Algorithm 1 on page 28 prescribe some condition that will generate a series of such point values).

Preprocessing the Build-MaxCR Input to Remove Redundant Views

The algorithm Build-MaxCR may be preceded by a preprocessing step as follows. If a view from \( V \) is not useful in answering \( Q \), that is, if no relational subgoal from this view
has the same predicate-name as any subgoal from $Q$, then such a view can be removed from $V$ before submitting it as input to Build-MaxCR. Thus, it is possible to decrease Build-MaxCR runtime by decreasing the number of views that are input to it.

**Using a Composite System with One Algorithm to Detect Single-Mapping MCRs and Another to Detect Multiple-Mapping MCRs**

The number of relational subgoals in a single-mapping CQAC MCR is at most equal to the number of relational subgoals in $Q$ [LMSS95] (this happens when each viewpoint in the rewriting covers only one relational subgoal of $Q$), but as we saw in Example 3.1 (page 19 in Section 3.1), the length of a multiple-mapping CQAC MCR may not be bounded (this happens when the available views are used repeatedly with different head-homomorphisms to generate a multiple-mapping MCR that is not contained in any other MCR of fewer subgoals).

Given a CQAC query $Q$ and a set of CQAC views $V$, it may be possible to use a composite system that uses a single-mapping algorithm such as [PH01] whenever it is known that the input $Q$ and $V$ will produce only single-mapping MCRs, but uses Build-MaxCR for the general case when it is known that the input $Q$ and $V$ may also produce multiple-mapping MCRs. If there are several inputs for which it is known that only single-mapping MCRs are possible (e.g., this is the case when the input query and views are all left or right semi-interval queries [ALM04]), then the performance of such a composite system, averaged across a number of executions, may be better than the corresponding performance of a system that uses Build-MaxCR alone on all inputs.

**Using a Brute-Force Approach for Finding Maximally Contained Rewritings for CQAC queries**

We begin with a review of the general idea of algorithm Build-MaxCR. Build-MaxCR finds the $k$-bounded UCQAC query $P$, which is the maximally contained rewriting of a CQAC query $Q$ using CQAC views $V$, as follows. Initially, it does a complete enumeration of the CQ parts $P_j$, of $k$-bounded CQAC queries defined on schema $V$. For each such $P_j$, it associates with $P_j$ a minimum set $S_j$ of inequality/nonequality ACs called summary

---

7A head homomorphism $h$ on a view $V$ is a mapping $h$ from the variables of $V$ to the variables of $V$. It maps each existential variable to itself, but may equate distinguished variables, that is, for every distinguished variable $X$, $h(X)$ is distinguished, and $h(X) = h(h(X))$. 

---
ACs, such that \( S'_j \) ensures the containment of \( P_j^{exp} \& S'_j \) in \( Q \). The output for Build-MaxCR is the union \( P \) of all the CQAC queries \( P_j \& S'_j \) for which the containment holds.

Thus, Build-MaxCR exhaustively enumerates all CQ parts (by considering all possible multisets up to size \( k \)). Also, it systematically considers all multisets of size 1, then all multisets of size 2, and so on, until it considers all multisets of size \( k \). A completely brute-force technique may not do a systematic size-wise enumeration of the multisets. Instead, it could be randomly generating a multiset of any size, and then discarding it \((i)\) if it is larger than size \( k \); or \((ii)\) if it is of size less than or equal to \( k \) but has already been processed earlier. Processing is faster for multisets of smaller sizes, hence Build-MaxCR considers these first. For a particular multiset, if there is a CQAC query resulting from that multiset that is an equivalent rewriting of \( Q \) using \( \mathcal{V} \), then Build-MaxCR detects this and immediately terminates all further processing. Hence, on an average, across multiple runs on a number of different inputs, the policy of considering first the multisets of smaller size enables Build-MaxCR to be more efficient in comparison to a completely brute-force way of considering the various multisets.

However, the main difference between Build-MaxCR and a brute-force approach lies in the finding of the AC part (rather than in the finding of the CQ part that was described earlier). The Build-MaxCR pseudocode (Algorithm 1 on page 28) and explanation (Section 3.3 on page 26) give the details of how AC parts are discovered in Build-MaxCR. Here, we outline how a completely brute-force approach would try to discover the possible AC parts for a given CQ part, so that the two parts conjoined together form a CQAC query that is contained in \( Q \).

Let there be a total of \( n_s \) symbols (variables/constants) in the expansion of the CQ part that is being considered. (If the query \( Q \) has additional constants then these constants are also included in determining the value of \( n_s \).) There are a total of “\( n_s \) choose 2” ways of picking 2 symbols out of these \( n_s \) symbols. For each such choice of a pair of symbols, say \( X \) and \( Y \), we can have a total of 6 possible ACs: \( X = Y \), \( X < Y \), \( X \leq Y \), \( X > Y \), \( X \geq Y \), and \( X \neq Y \). Thus, the total possible number \( p \) of ACs is

\[
p = 6 \times \frac{n_s!}{(n_s - 2)!2!}
\]
that is,
\[ p = \frac{3n_s!}{(n_s - 2)!} \]

The brute-force approach would have to consider all possible subsets of this set of \( p \) possible ACs. There are \( 2^p \) such subsets. For each of these subsets, the brute-force approach would have to check whether the CQAC query formed by taking a conjunction of the ACs in the subset with the CQ part being considered is contained in the query \( Q \). And for this it would have to perform the expensive CQAC containment test (Section 2.1.2, page 14). That is, it would have to perform one test for each of the \( 2^p \) subsets. If \( t_{cqac\text{-}containment} \) is the time taken for a single CQAC containment test, then the total time taken by the brute-force approach for processing just one CQ part is

\[ t_{cqac\text{-}containment} \times 2^{\frac{3n_s}{(n_s - 2)}} \]

that is,

\[ t_{cqac\text{-}containment} \times 8^{\frac{n_s}{(n_s - 2)}} \]

If the brute-force method was used, this cost would have to be incurred each time for each CQAC-rewriting template that is being examined. Thus, in general, a brute-force checking of all possible ACs that can be used to form an MCR would be prohibitively expensive. In contrast, recall that for this purpose the algorithm Build-MaxCR does not need to perform any containment test at all.

**Finding Equivalent Rewritings of CQAC queries**

Having seen a naive brute-force method for finding contained rewritings (please see Section 3.6 page 48), next, we examine a different rather sophisticated strategy for finding rewritings of CQAC queries — although this strategy applies to only the special case of finding equivalent rewritings. Since an equivalent rewriting is a special case of a contained rewriting, we present a brief comparison of the algorithm in [ACGP06] with Build-MaxCR. (Please see page 8 in the Related Work section in Chapter 1 for a discussion on how the work in [ACGP06] is related to the work in this thesis.) However, note that the search space for finding an equivalent rewriting is significantly more restricted as compared to the search space for finding a contained or a containing rewriting. For example, if a view \( V \) in the input viewset \( \mathcal{V} \) has a \( p \)-subgoal but the input query \( Q \) has no \( p \)-subgoal, then
it is already known that $V$ can never be useful in forming an equivalent rewriting of $Q$. Thus, some simple preprocessing step such as this one could itself allow us to eliminate from consideration perhaps a large number of irrelevant views. Note that the condition tested during such a preprocessing is much more stringent as compared to the possible preprocessing condition for Build-MaxCR (described on page 47 in Section 3.6). Thus, after the preprocessing step for finding equivalent rewritings, we would be left with a much smaller viewset that would need to be processed. The reduced size of the input viewset will allow an equivalent-rewriting algorithm to perform a comparatively faster processing.

Broadly speaking, for each total-order CQAC component $Q_i$ of the input query $Q$, the algorithm in [ACGP06] first uses chase-like techniques to find a containing rewriting of $Q_i$. It then uses the standard canonical databases test (please see Section 2.1.2 on page 14 for a description) to check whether this containing rewriting also happens to be a contained rewriting, and if so it concludes that this rewriting is an equivalent rewriting. The union of the equivalent rewritings for all $Q_i$’s is returned as the UCQAC answer of the algorithm. The algorithm performs optimizations such as including in its intermediate containing rewriting, only those tuples (subgoals) that overlap with the MCDs (from algorithm MiniCon of [PH01]). This gives a containing rewriting that is more likely to also be an equivalent rewriting, and if it is not, then at least it provides an early termination condition, which can help improve the efficiency of the algorithm.

In contrast, algorithm Build-MaxCR finds the CQAC elements of its UCQAC answer as follows. It tries all possible multisets of viewheads to construct the relational part of the rewriting. The ACs are obtained by examining all possible total orders and then taking the logical OR of exactly those total orders for which the candidate relational part is contained in the query.

For comparison of the algorithm in [ACGP06] with Build-MaxCR, note that by the time the algorithm in [ACGP06] comes to the part where it has to ensure that the rewriting is a contained one, it already knows that the CQAC query, say $A_i$, which is being tested for containment in $Q_i$, is a containing rewriting of $Q_i$, and that it has to be kept

---

8 Any CQAC query $Q$ can be expressed as $Q = \bigcup_i Q_i$, where each $Q_i$ is a total-order CQAC component of $Q$. That is, the relational part of $Q_i$ is the same as the relational part of $Q$ and the ACs of $Q_i$ define a total order on the variables and constants in $Q$.

9 This can be thought of as chasing [DNR08] $Q$ with the forward constraints [DLN05] obtained from the views in $V$. 
that way (i.e., it has to remain containing). Thus, when the \textit{contained-rewriting part} of the algorithm in \cite{ACGP06} is processing \(A_i\), it already has the containing rewriting \(A_i\) (that we refer to as the “pre-built” \(A_i\)). Thus, the (pre-built) AC part of \(A_i\) as well as the (pre-built) relational part of \(A_i\) are both already available. For any variable \(X\) in \(A_i\), if the \(A_i\) output by the initial containing-rewriting part of the algorithm in \cite{ACGP06} does not have ACs restricting the values that \(X\) can take, it is either (\(i\)) because there was no need to add ACs on \(X\) to ensure that \(P\) is the tightest possible fit for \(Q\); or (\(ii\)) because \(X\) is not present in the head of any view that contributes it, so there is no way to access \(X\) in order to impose a restricting AC on it. Thus, there is no need for the contained-rewriting part of the algorithm in \cite{ACGP06} to add any ACs. That is, the requirement that the rewriting also has to be a containing one, gives the algorithm in \cite{ACGP06} a considerably more restricted search space as compared to algorithm Build-MaxCR. This is true for the pre-built ACs as well as the pre-built relational part consisting of the view tuples that the algorithm in \cite{ACGP06} obtains from its initial containing-rewriting part.

On the other hand, algorithm Build-MaxCR does not have any pre-built \(A_i\) and has to start from scratch in discovering ACs. If it used a brute-force approach to try all possible ACs and each time used a CQAC containment test to check whether containment holds, then the cost would be prohibitive, as discussed earlier (page \pageref{sec:cost}) in this section. But the algorithm Build-MaxCR uses its strategy of constructing summary ACs, thus bypassing the need to perform any explicit CQAC containment checking, and still maintains the soundness and completeness of the algorithm. In contrast, the algorithm in \cite{ACGP06} does need to perform CQAC containment tests. However, since it can restrict its attention to only the select number of ready-made \(A_i\)’s that have been output by its initial containing-rewriting procedure, the total cost seems more affordable in comparison to the cost that it would have had to incur in performing containment tests for the entire unrestricted search space. Also, note that in the algorithm in \cite{ACGP06}, each CQAC element of the UCQAC answer corresponds to one total order of \(Q\), whereas this need not be the case in algorithm Build-MaxCR. The ability to fuse multiple total-order CQAC queries into a single CQAC query, allows an algorithm (such as Build-MaxCR) to reduce the total number of CQAC components in its final UCQAC answer.
3.6.1 Complexity of Algorithm Build-MaxCR

We now turn to a discussion on the complexity of algorithm Build-MaxCR (Algorithm 1 on page 28). Consider a Build-MaxCR problem input \((Q, V, k)\). Suppose that there are \(n\) views in \(V\), and that each of these views and the query \(Q\) contain a maximum of \(r\) relational subgoals, with \(p\) being the maximum arity of any subgoal. \(p\) can be treated as a constant (assuming fixed database schema).

For any given \(t\), the number of times that the outermost \(t\) loop (line 2 of Algorithm 1 on page 28) is executed is “\((n + t - 1)\) choose \(t\)”, represented as \(\binom{n+t-1}{t}\), and is equal to
\[
\frac{(n + t - 1)!}{(n - 1)!t!}
\]
This is the total possible number of ways in which we can form a multiset of size \(t\) using elements from a pool of size \(n\) (the \(n\) views in this case).

The next inner for-loop (line 10 in Algorithm 1 on page 28), will execute \(|M|\) number of times, where \(M\) is the set of all mappings from the relational subgoals of \(Q\) to the same-name subgoals in the expansion of the CQAC-rewriting template being considered by Build-MaxCR. The CQAC-rewriting template being considered consists of \(t\) viewheads. On expansion, each viewhead contributes at most \(r\) relational subgoals. Thus, the expansion contains at most \(t \times r\) relational subgoals. Consider the worst case where all subgoals of \(Q\), as well as each subgoal of each view, has the same predicate name. In this case, the first relational subgoal of \(Q\) can map to any one of the \(t \times r\) relational subgoals in the expansion. Also, the second relational subgoal of \(Q\) can map to any one of the same \(t \times r\) relational subgoals in the expansion. The same is true for the third, fourth, \ldots, \(r\)th subgoal of \(Q\). Thus, each of the \(r\) subgoals of \(Q\) can map to any of the \(t \times r\) subgoals in the expansion, for a total of \((t \times r)^r\) mappings. Hence, that loop (line 10 in Algorithm 1 on page 28) will execute at most \((t \times r)^r\) times.

Producing and processing the set \(D_t\) of canonical databases (lines 6 and 11 of Algorithm 1 on page 28) takes time proportional to the total possible number of MaxCR canonical databases for the CQAC-rewriting template being considered. There are \(t \times r\) relational subgoals in the expansion of the CQAC-rewriting template. Each relational subgoal can contribute at most \(p\) variables or constants (to the set of symbols used to form MaxCR canonical databases for the CQAC-rewriting template being considered). Hence the total
possible number of variables and constants contributed (to the set of symbols used to form MaxCR canonical databases for the CQAC-rewriting template being considered) by the relational part of the expansion of the CQAC-rewriting template is \( t \times r \times p \). The symbols included in forming MaxCR canonical databases include all the variables and constants in the expansion of the CQAC-rewriting template plus the constants in \( Q \). Assuming that the number of constants contributed by the ACs in the expansion of the CQAC-rewriting template, and by \( Q \), is a constant, and also assuming that \( p \) is a constant, the total possible number of symbols on which Build-MaxCR forms MaxCR canonical databases is \( O(t \times r) \).

There are \((t \times r)!\) permutations on \( t \times r \) symbols. For any given permutation of \( t \times r \) symbols, each of the \((t \times r) - 1\) gaps between these symbols can hold one of the two signs “<” or “=” in order to form a total order on those symbols. Thus, for a given permutation there are \(2^{((t \times r) - 1)}\) total orders. Therefore, across all permutations, there are at most

\[ ((t \times r)!) \times 2^{((t \times r) - 1)} \]

possible total orders. Hence, this is the total possible number of MaxCR canonical databases that may have to be considered for each given CQAC-rewriting template and for each mapping.

Hence, for a given \( t \), the total time taken by Build-MaxCR can be represented by

\[
\frac{(n + t - 1)!}{(n - 1)!t!} \times (t \times r)^r \times ((t \times r)!) \times 2^{((t \times r) - 1)}
\]

As \( t \) ranges from 1 to the user-specified \( k \), the total possible time taken by Build-MaxCR is

\[
\sum_{t=1}^{k} \frac{(n + t - 1)!}{(n - 1)!t!} \times (t \times r)^r \times ((t \times r)!) \times 2^{((t \times r) - 1)}
\]

In the worst case, each mapping in \( \mathcal{M} \) gives rise to one CQAC element in the final UCQAC answer \( P \) of Build-MaxCR. Suppose that there are no multiple-mapping rewritings. (When two or more mappings collectively produce one single CQAC component in \( P \), then the total number of CQAC components that are present in \( P \) are fewer than the total number CQAC components that would have been present if each of those mappings had produced a separate CQAC component. Also, in practice, several of the mappings may not produce a CQAC component in \( P \) at all, but in this worst-case scenario, we have assumed that each mapping does produce a CQAC component.) Hence for a given CQAC-rewriting
template there are a total of $|\mathcal{M}|$ CQAC components that get added to $P$. Thus for a given $t$, there will be

$$\frac{(n+t-1)!}{(n-1)!t!} \times (t \times r)^r$$

such CQAC components. Across all values of $t$, therefore

$$z = \sum_{t=1}^{k} \frac{(n+t-1)!}{(n-1)!t!} \times (t \times r)^r$$

is the total possible number of CQAC queries in the UCQAC output $P$ of Build-MaxCR. (But in practice, this number is usually much smaller.) Also, another important point about the time complexity of Build-MaxCR is that it is able to save time during its processing, by completely bypassing any explicit containment checking. Thus, it does not have to incur the cost $t_{\text{cqac-containment}}$ of even a single CQAC containment test.

### 3.7 Correctness of Algorithm Build-MaxCR

We now formulate theorems that establish soundness and completeness of Build-MaxCR, as well as the decidability results for two decision versions of the problem (page 65) of constructing UCQAC $k$-bounded MCRs for CQAC queries using CQAC views. In Section 3.7.1 (page 55), we include theoretical results on “total-order CQAC queries”. These are 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 correctness of Build-MaxCR. In Section 3.7.2 (page 60) we prove the soundness of Build-MaxCR, and in Section 3.7.3 (page 62) we prove its completeness.

#### 3.7.1 Total-Order CQAC Queries

Recall that Build-MaxCR does not have to do containment checking, either during the construction of the outputs or (as final containment test) to test that the Build-MaxCR outputs are contained in the input query. Also note that in the formal statements below, for a Build-MaxCR problem input $(Q, \mathcal{V}, k)$, $P$ denotes a CQAC-rewriting template (Definition 3.6 on page 24 in Section 3.2) 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 3.1.** 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 \( P' : \neg P_{\exp} \& S \) has exactly one tuple \( t \) in the answer on \( D \), regardless of the choice of head variables of \( P' \). Further, \( t \) can be obtained by applying to the head variables of \( P \) the generative assignment mapping (see Definition 3.8 on page 24 in Section 3.2) from \( P_{\exp} \) to \( D \).

Note that the generative assignment mapping is not the only assignment mapping that can be used to obtain the answer tuple to \((P')_{\exp}\) on \( D \) in the context of Proposition 3.1.

**Proof.** (Proposition 3.1) 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 3.8 (page 24 in Section 3.2) \( M \) is indeed the generative assignment mapping from \( P_{\exp} \) to \( D \).

**Proposition 3.2.** 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 query \( P' \) has an empty answer on \( D \) whenever \( S' \) enforces \( m' > m \) distinct values on the variables of \( P' \).

The proof of Proposition 3.2 is an immediate variation on the observations in the proof of Proposition 3.1 (page 56).

Note that when \( m' < m \), in the terminology of Proposition 3.2, and even when \( m' = m \) (while \( S' \) is still not the same as \( S \)), then it is also possible that we get an empty

\[^{10}\text{Recall that all variables of both } P \text{ and } P_{\exp} \text{ are distinguished, and that the sets of body variables of } P \text{ and } P_{\exp} \text{ are the same.}\]
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 these points. In Example 3.13, the total order that defines query $P'$ 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 3.14 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 3.13.** Consider a CQAC-rewriting template $P$ defined in terms of a single view $V$

\[
\begin{align*}
P(X, Y) : & \ - V(X, Y). \\
V(X, Y) : & \ - s(X, Y), s(Y, X).
\end{align*}
\]

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

\[
\begin{align*}
P(X, Y, Z, T) : & \ - V(X, Y, Z, T). \\
V(X, Y, Z, T) : & \ - s(X, Y), s(Y, Z), s(Z, T).
\end{align*}
\]

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 $P^{\text{exp}}_S(X = Y = Z = T)$.

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 3.3.** 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 $P^{\text{exp}}_S S'$, 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 3.3 for this case holds by Proposition 3.1 (page 56).

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 3.1 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 with respect to total order $S'$ to the smallest-value variable with respect to 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 3.13 (page 57), 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 3.4. 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 \(P_{\text{exp}} \& S'\), 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 \(\lambda\) from \(P'\) to \(D\), such that \(t\) can be obtained by using \(\lambda\). Let \(\hat{\lambda}\) be the mapping from the relational subgoals of \(P'\) to the stored tuples of \(D\), such that \(\lambda\) is induced by \(\hat{\lambda}\). 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 \(\hat{\lambda}\), between the variables of \(P'\) and the \(m'\) distinct values in the tuples of \(T\). (See proof of Proposition 3.1 on page 56 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 \(\hat{\lambda}\) is a generative assignment mapping from \(P'\) to the database \(T\).

Corollary 3.1. In the setting of Proposition 3.4, the answer to \((P')_{\text{exp}}\) on \(D'\) has exactly one answer tuple, under set semantics for query evaluation.

The claim of Corollary 3.1 follows immediately from Proposition 3.1 (page 56) and from the construction of \(D'\) as outlined in the proof of Proposition 3.4 (page 59).

Proposition 3.5. 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 \(D\). 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 $D$ with respect to $P'$. 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 $D$.

### 3.7.2 Soundness of Build-MaxCR

In this subsection, we give the result for the soundness of algorithm Build-MaxCR followed by the corresponding proof, which uses results from Section 3.7.1 (page 55) and the following Lemma 3.1.

**Lemma 3.1.** Given input $(Q, V, k)$ 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.

**Theorem 3.1.** (Soundness of Build-MaxCR) For a Build-MaxCR problem input $(Q, V, k)$, 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 $Q$. (Here, $S$ is a conjunction of ACs.)
Proof. 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 3.1 \((P')^{\exp}\) is equivalent to the union \(\bigcup_i P'_i\), 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 each such \(i\), 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 3.4 on page 22 in Section 3.2 — 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 3.1 (page 56) 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 3.3 (page 58) 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 3.3 (page 58) 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 3.2 (page 56), the answer to \( P'_i \) on \( D \) is empty;

(d) \( m > m_i \); then, by Proposition 3.4 (page 59), 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 3.4 (page 59) and from Corollary 3.1 (page 59) 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.

\[ \square \]

### 3.7.3 Completeness of Build-MaxCR

In this section, we provide the statement and the proof for the completeness of algorithm Build-MaxCR.

**Theorem 3.2. (Completeness of Build-MaxCR)** For a Build-MaxCR problem input \((Q, \mathcal{V}, k)\), let \( R \) be a UCQAC query defined in terms of \( \mathcal{V} \), such that (i) in each CQAC component \( R_i \) of \( R \), the number of relational subgoals of \( R_i \) does not exceed \( k \), and (ii) \( R^{exp} \subseteq Q \). Then (1) the output of Build-MaxCR is not empty, and (2) denoting by \( P \) the UCQAC output of Build-MaxCR, we have that \( R^{exp} \sqsubseteq P^{exp} \).

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

We “expand” the UCQAC query \( R \) into its CQAC components, as \( R = \bigcup_{i=1}^{m} R_i \) for some natural number \( m \). Here, each \( R_i \) is a CQAC component of \( R \). By \( t \) being a tuple

\[ \text{12Thus } P_r \text{ is guaranteed to have at most } k \text{ relational subgoals.} \]
in the answer to $R^{\exp}$ on $D$, there exists a CQAC query $R_i$ in $R$, such that $t \in R_i^{\exp}(D)$. (In case more than one CQAC query in $R^{\exp}$ returns $t$ on $D$, we choose an arbitrary such CQAC query $R_i^{\exp}$.) We represent $R_i^{\exp}$ as a union of total-order CQAC queries, where each total order is on all the variables and constants of $R_i^{\exp}$: $R_i^{\exp} = \bigcup_{j=1}^{l} R^*_{j} \& S_{j}$, for some natural number $l$. Here, each $R^*_{j} \& S_{j}$ is a CQAC query 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 CQAC queries in the union $R_i^{\exp} = \bigcup_{j=1}^{l} R^*_{j} \& S_{j}$.

By $t \in R_i^{\exp}(D)$, there must exist an $S_{j}$ (for some $j \in \{1, \ldots, l\}$) such that $t \in (R^*_{j} \& S_{j})(D)$. We denote $R^*_{j} \& S_{j}$, 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 $R^*$ with respect to $D$, 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 $D$) 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 query $P_r$ such that $t \in P_r^{\exp}(D)$.

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

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 $S''_{j} = S_{j} - S^\prime_{j}$, with “−” understood as set difference, are either comparisons of variables of $R^*$ with those constants that do not occur in the expansion of $V_{(n)}$, 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 $V_{(n)}$. This observation follows from our assumption that all (U)CQAC
queries that we consider in this thesis 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 on total-order CQAC queries in Section 3.7.1 on page 55.) 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 3.7.1 (page 55) 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 $R^{exp}$, 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 $(Q, V, k)$.

We have seen that on each database $D$ on which $R^{exp}$ produces an answer, expansions of some CQAC queries considered by Build-MaxCR also produce exactly the answers to $R^{exp}(D)$. Thus, to show containment of $R^{exp}$ in the expansion of the output of Build-MaxCR for the input $(Q, V, k)$, it remains to show that for all the (considered above) CQAC queries $P_r$ that “cover” $R^{exp}$ 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 $P^{exp}_r$) 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 query in $R$. (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 $\mathcal{V}_n$ used to build $P_r$. (That is, $P_r$ is defined as a conjunction of $\mathcal{V}_n$ 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 $R^\text{exp}$ 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 CQAC query $P'_r$ 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 query $R_i$ in $R$, there must exist a CQAC query $P_j$ in the output $\mathcal{P}$ of Build-MaxCR (that is, $\mathcal{P} = \bigcup_{j=1}^z P_j$ 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 $R^\text{exp} \sqsubseteq P^\text{exp}$ as required.

By Theorem 3.1 (page 60) and Theorem 3.2 (page 62) we obtain immediately the following two results:

**Theorem 3.3. (Decidability)** Given a CQAC query $Q$, a set $\mathcal{V}$ of CQAC views, and a natural number $k$. (1) It is decidable to determine whether $Q$ has a UCQAC $k$-bounded contained rewriting in terms of $\mathcal{V}$. (2) Further, given in addition a UCQAC $k$-bounded query $R$ defined in terms of $\mathcal{V}$, the problem of determining whether $R$ is a UCQAC $k$-bounded MCR for $Q$ using $\mathcal{V}$ is decidable.

Thus, these results resolve in the positive the problem of decidability of the existence of a UCQAC size-limited MCR for CQAC queries and views.
Chapter 4

Containing Rewritings

In this chapter, we turn to \textit{minimally containing rewritings} \cite{GM99,DLN07,CCM07}, which we abbreviate as MiCRs. 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. We start by presenting a definition of a MiCR in Section 4.1 (page 67). In Section 4.2 (page 67), we introduce the notion of a \textit{minimized MiCR}, which formalizes the above efficiency intuition. The main contribution of this chapter is an algorithm that we call \textit{pruned-MiCR}, see Section 4.3 (page 69). Given a CQAC MiCR for a given problem input (i.e., for a CQAC query and a set of CQAC views), pruned-MiCR \textit{globally} minimizes the MiCR in an \textit{efficient} and \textit{scalable} way. We also outline algorithm CB-MiCR (Section 4.3 on page 69), which uses an exhaustive strategy to find globally minimized MiCRs. We give proofs for the correctness of the MiCR algorithms (Section 4.4 on page 84), and conclude with the discussion on a technique that can be used to reduce the time taken for CQAC containment checking, by first partitioning ACs according to variable types (Section 4.5 on page 88). Our experimental results in Chapter 5 (page 93) 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.
4.1 Minimally Containing Rewritings

The word “minimal” in “MiCR” refers to a containing rewriting that contains the fewest false positives (in the given rewriting language) with respect to the query answer. Thus, an algorithm trying to find a containing rewriting of a CQAC query \( Q \) using a set of CQAC views \( V \), can try to minimize the false positives that would be obtained by running the rewriting, by constructing a rewriting that is minimally containing.

**Definition 4.1.** (Minimally containing rewriting) A query \( R \) defined in query language \( L_1 \) is a minimally containing rewriting (MiCR) of a query \( Q \) defined in language \( L_2 \) using a set of views \( V \) defined in language \( L_3 \) if: (1) \( R \) is a containing rewriting of \( Q \) in terms of \( V \), and (2) there exists no containing rewriting (in language \( L_1 \)) \( R' \) of \( Q \) using \( V \), such that the expansion of \( R' \) is properly contained in the expansion of \( R \).

For the results in this chapter, each of \( L_1 \) through \( L_3 \) is the language of CQAC queries.

Algorithm pruned-MiCR employs a novel strategy that makes use of constructs that we refer to as “buckets”. (Please see Section 4.3.2 on page 73 for details.) Pruned-MiCR uses its buckets to minimize (with respect to the given viewset) the number of relational subgoals in a given MiCR. In a CQAC query with \( n \) relational subgoals, there are \( n - 1 \) joins — so decreasing the number of relational subgoals in a CQAC query decreases the number of joins that have to be computed when that CQAC query is being evaluated. In practical applications, typically, a MiCR may be computed once and then executed repeatedly (possibly with different parameters for different runs). 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 4.3 on page 75 in Section 4.3.4), and thus many fewer joins, such a performance improvement would have a significant payoff.

4.2 Preliminaries for the MiCR Algorithms

Section 4.1 (page 67) provided a definition of MiCRs. 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 4.2. (Minimized MiCR) Given a CQAC query \( Q \) and a set of CQAC views \( V \), CQAC MiCR \( R \) of \( Q \) using \( V \) is a minimized (CQAC) MiCR of \( Q \) using \( V \) if removing any relational subgoal of \( R \) results in query \( R' \) such that \( R \) and \( R' \) are not equivalent as expansions, that is \( R^{\text{exp}} \neq (R')^{\text{exp}} \).

By definition, if we delete even a single relational subgoal from a minimized MiCR, it no longer remains a MiCR. We now consider the notion of a “globally minimal” minimized MiCR. A globally minimal minimized CQAC MiCR for a CQAC query \( Q \) and set \( V \) of CQAC views has the minimum number of relational subgoals among all CQAC queries defined using \( V \) that are equivalent (as expansions) to a given CQAC MiCR for \( Q \) and \( V \). It turns out that a globally minimal minimized MiCR (i.e., a minimal MiCR) may not be unique for a given \((Q, V)\), as shown by the following example.

**EXAMPLE 4.1.** 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 the following rewriting:

\[
R() \ :- \ V_1(), V_2().
\]

This MiCR \( R \) (for \( Q \) and \( \{V_1, V_2\} \)) is equivalent as expansions to 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 \((Q, \{V_1, V_2\})\).

Next, we give an example that 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 whenever algorithm pruned-MiCR and algorithm CB-MiCR output a minimized MiCR, it is guaranteed to be a globally minimized MiCR, see Section 4.4 (page 84) for the details.
**EXAMPLE 4.2.** Consider a Boolean CQ query $Q$ and Boolean CQ views $V_1$, $V_2$, $V_3$, $V_4$, and $V_5$.

\[ Q() \, :: \, p(), s(), t(), u(). \]
\[ V_1() \, :: \, p(), s(). \]
\[ V_2() \, :: \, t(), u(). \]
\[ V_3() \, :: \, s(), t(). \]
\[ V_4() \, :: \, p(). \]
\[ V_5() \, :: \, u(). \]

\[ R() \, :: \, V_1(), V_2(), V_3(), V_4(), V_5(). \]

\[ R'() \, :: \, V_1(), V_2(). \]
\[ R''() \, :: \, V_3(), V_4(), V_5(). \]

$R$ is the full MiCR for $(Q, \{V_1, V_2, V_3, V_4, V_5\})$. $R'$ and $R''$ are two distinct minimized MiCRs for $R$. $R'$ has two relational subgoals and $R''$ has three relational subgoals. This shows that two distinct minimized MiCRs for a given CQAC MiCR can have different numbers of relational subgoals. Note that algorithms pruned-MiCR and CB-MiCR will output $R'$ (which is globally minimal) as opposed to $R''$ (which is a minimized MiCR but not a globally minimal minimized MiCR).

\[
\square
\]

### 4.3 Description of the MiCR Algorithms

In this section, we discuss in detail the working of algorithm full-MiCR (Section 4.3.1 on page 71), algorithm pruned-MiCR (Section 4.3.2 on page 73), and algorithm CB-MiCR (Section 4.3.3 on page 74). The bucket-construction procedure used by algorithm pruned-MiCR is described further through the pseudocode in Algorithm 4 on page 72 and through Example 4.3 (page 75) in Section 4.3.4. This is followed by discussions on the incompleteness (Section 4.3.5 on page 78) and complexity (Section 4.3.6 on page 84) of pruned-MiCR.
Algorithm 3: The algorithm full-MiCR.

Input: CQAC query $Q$; set of CQAC views $V$

Output: the full MiCR $R$ of $Q$ in terms of $V$

1 Initialize $R$ to have a head that is identical to the head of $Q$ and a body that is empty;

2 for each view $V$ in $V$ do

3 for each containment mapping $\mu$ from the relational subgoals in $V$ to the relational subgoals in $Q$ do

4 construct $h(V)$ by replacing each distinguished variable $X$ in $V$ by $\mu(X)$;

5 $ac \leftarrow \text{null}$;

6 for each AC $ac_i \in \text{closure}(AC(Q))$ do

7 if all variables in $ac_i$ appear in $h(V)$ then

8 $ac \leftarrow ac \land ac_i$;

9 if $AC(Q) \Rightarrow \mu(AC(h(V)))$ then

10 conjoin $h(V), ac$ with the existing body of $R$;

11 if there is some variable in the head of $Q$ that is not present in any relational subgoal in $R$ then

12 output that there exists no safe MiCR of $Q$ using $V$ and stop;

13 Output $R$ as the full MiCR;
4.3.1 The Algorithm Full-MiCR

The algorithm full-MiCR (Algorithm 3) accepts on input a CQAC query Q, and a set of CQAC views V, and returns CQAC rewriting R defined in terms of V, such that R is a MiCR of Q using V. The output R of algorithm full-MiCR is given as input to algorithm pruned-MiCR (Algorithm 4 on page 72) as well as to algorithm CB-MiCR (Algorithm 5 on page 74).

The algorithm full-MiCR starts by initializing R, so that the head of R is identical to the head of Q, and the body of R contains no subgoals. Then it examines one by one each view V in V. For each containment mapping µ from the relational subgoals in V to the relational subgoals in Q, full-MiCR constructs h(V) from V, where the function (homomorphism) h is such that it replaces each distinguished variable X in V, by the image µ(X) of X under mapping µ. The algorithm then constructs ac, which is a conjunction of some of the ACs from closure(AC(Q)), where closure(AC(Q)) is the result of taking the closure [Ull89] of the ACs in Q. An AC from closure(AC(Q)) is included in ac if and only if all variables in that AC appear in h(V). The algorithm then checks AC(Q) ⇒ µ(AC(h(V))), where “⇒” is used as the symbol for logical implication. That is, full-MiCR checks if the ACs of Q imply the ACs of h(V). If this is true, it conjoins the head of h(V) and all 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 query R is output by algorithm full-MiCR. We call R the full MiCR of Q using V. Otherwise, the algorithm declares that there exists no safe containing rewriting of Q using V, and stops.

The algorithm full-MiCR is sound and complete for all problem inputs for which the homomorphism property (Definition 3.9 on page 25 in Section 3.2) holds between the expansion of the rewriting and the input query. Section 4.4 (page 84) gives the soundness and completeness proofs for algorithm full-MiCR. (The problem of finding the full MiCR has also been addressed in [DLN05], please see the discussion on related work on page 8 in Chapter 1.)
Algorithm 4: The algorithm pruned-MiCR.

**Input**: CQAC query \( Q \); set of CQAC views \( V \), the full MiCR \( R \) of \( Q \) in terms of \( V \) output by algorithm full-MiCR

**Output**: the full MiCR \( R \) of \( Q \) in terms of \( V \), globally minimal minimized MiCR \( R' \)

1 //The algorithm full-MiCR (Algorithm 3 on page 70) is first executed to obtain the full MiCR \( R \). This \( R \) is then input to algorithm pruned-MiCR that constructs buckets for approximate containment checking to find the minimal MiCR \( R' \).

2 for each relational subgoal \( g_r \) in each \( h(V) \) whose head is included in \( R \) do

3     for each subgoal \( g_q \) in \( Q \) such that \( g_r \) maps to \( g_q \) do

4         \( V' \leftarrow h(V), ac; \) constructNewBucket ← true;

5         for every bucket \( (g_q, g_{r_i}) \) that already exists for \( g_q \) do

6             if \( g_{r_i} \) is more restrictive than \( g_r \) then

7                 constructNewBucket ← false;

8             if \( g_r \) is more restrictive than \( g_{r_i} \) then

9                 delete the bucket \( (g_q, g_{r_i}) \);

10                if \( g_{r_i} \) and \( g_r \) are equally restrictive then

11                   insert \( V' \) in the existing bucket \( (g_q, g_{r_i}) \);

12                   constructNewBucket ← false;

13                if constructNewBucket is true then

14                   create a new bucket \( (g_q, g_r) \) with one entry \( V' \) in it;

15 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;

16 Construct CQAC query \( R' \) by taking a conjunction of all the selected \( V' \) entries;

17 if \( (R')^{exp} \subseteq R^{exp} \) then

18     Output \( R \) as the full MiCR and \( R' \) as a globally minimal minimized MiCR;

19 else

20     Output \( R \) as the full MiCR;
4.3.2 The Algorithm Pruned-MiCR

The algorithm pruned-MiCR takes as input a CQAC query $Q$, a set of CQAC views $V$, and the CQAC $R$ which is a full MiCR of $Q$ using $V$. Any algorithm that finds the full MiCR of $Q$ using $V$ (e.g., the algorithm from [DLN05]) can be used to provide the $R$ that is input to algorithm pruned-MiCR. In our experiments (Section 5.2 on page 97), we use the output of algorithm full-MiCR (described in Section 4.3.1 on page 71) as the input to algorithm pruned-MiCR. Algorithm pruned-MiCR returns CQAC queries $R$ and $R'$ defined in terms of $V$, such that $R$ is the MiCR that was input to it and $R'$ is a globally minimal minimized MiCR of $Q$ using $V$. Pruned-MiCR may not always be able to find $R'$ in which case it returns only the full MiCR $R$. Algorithm 4 (page 72) gives the pseudocode for pruned-MiCR and Example 4.3 (page 75) illustrates its bucket-forming strategy.

Algorithm pruned-MiCR starts with the full MiCR $R$ and applies its bucket-forming strategy. For every relational subgoal $g_r$ in the body of every $h(V)$ whose head is included in $R$, and for every subgoal $g_q$ in $Q$ such that $g_r$ maps to $g_q$, pruned-MiCR tries to insert $V'$, which is $h(V)$ conjoined with the ac for that $h(V)$, into appropriate buckets as follows. It compares $g_r$ with $g_{r'}$, for all $g_{r'}$’s for which it has previously constructed a bucket $(g_q, g_{r'})$ involving the same query subgoal $g_q$. If any $g_{r'}$ is strictly more restrictive than $g_r$ (i.e., if $g_r$ can map to $g_{r'}$ in a containment mapping), then the flag $constructNewBucket$ for subgoal $g_r$ (that was initialized to true), is now made false. However, if instead for any $g_{r'}$, it finds that $g_r$ is strictly more restrictive than $g_{r'}$, then the bucket $(g_q, g_{r'})$ for that $g_{r'}$ is deleted. Finally, if there exists some $g_{r'}$ that is exactly as restrictive as $g_r$, then $V'$ is inserted into the existing bucket $(g_q, g_{r'})$ and flag $constructNewBucket$ is made false[1]. 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 [CLRS01] 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. Let $R'$ be a CQAC query whose body is the conjunction of exactly those $V'$’s that have been selected by the minimum-set-cover algorithm and whose head is identical to the head of $R$. If $R'$ is safe and its expansion is contained in the expan-

---

[1] However, if the head of the $h(V)$ that corresponds to $g_r$ contains at least one head variable that is not present in the head of any $h(V)$ that corresponds to $g_{r'}$, then $constructNewBucket$ is not made false, and $V'$ is not inserted into that bucket.
4.3.3 The Algorithm CB-MiCR

Algorithm 5: The algorithm CB-MiCR.

**Input**: CQAC query \( Q \); set of CQAC views \( \mathcal{V} \), the full MiCR \( R \) of \( Q \) in terms of \( \mathcal{V} \) output by algorithm full-MiCR

**Output**: the full MiCR \( R \) of \( Q \) in terms of \( \mathcal{V} \), globally minimal minimized MiCR \( R' \)

1. //The algorithm full-MiCR (Algorithm 3 on page 70) is first executed to obtain the full MiCR \( R \). This \( R \) is then input to algorithm CB-MiCR that performs an exhaustive search for the minimal MiCR \( R' \).
2. \( n_R \leftarrow \) number of \( V' \)’s in \( R \);
3. **for** \( t = 1 \) to \( n_R \) **do**
4. \( U_t \leftarrow \) set of all \( t \)-sized subsets of the \( V' \)’s in \( R \);
5. **while** \( U_t \neq \emptyset \) **do**
6. \( \mathcal{R}_t \leftarrow \) one subset from \( U_t \); \( U_t \leftarrow U_t - \{\mathcal{R}_t\} \);
7. \( R_t \leftarrow \) the CQAC query formed by taking the conjunction of all the subgoals in all \( V' \)’s in subset \( \mathcal{R}_t \);
8. **if** \( R_t^{exp} \sqsubseteq R^{exp} \) **then**
9. \( R' \leftarrow R_t \);
10. Output \( R \) as the full MiCR and \( R' \) as a globally minimal minimized MiCR and stop;

Given a full MiCR \( R \), algorithm pruned-MiCR outputs \( R \) on all (CQAC) inputs and is also able to find a globally minimal minimized MiCR \( R' \) on some inputs. In order to have an algorithm which has the same inputs/outputs as pruned-MiCR but which also finds a globally minimal minimized MiCR on all inputs, we outline algorithm CB-MiCR, which does a simple exhaustive search of all subsets of the set of relational subgoals in the full MiCR \( R \) that is input to it. The obvious tradeoff involved is that CB-MiCR typically runs slower than pruned-MiCR. Our experimental results presented in Section 5.2 (page 97)
confirm this observation.

Both algorithms CB-MiCR and pruned-MiCR, take as inputs a CQAC query \( Q \), a set of CQAC views \( V \), and the CQAC \( R \) which is a full MiCR of \( Q \) using \( V \). Any algorithm that finds the full MiCR of \( Q \) using \( V \) (e.g., the algorithm from [DLN05]) can be used to provide the \( R \) that is input to algorithm CB-MiCR. In our experiments (Section 5.2 on page 97), we use the output of algorithm full-MiCR (described in Section 4.3.1 on page 71) as the input to algorithm CB-MiCR. CB-MiCR forms candidates by looking at all possible subsets of the set of relational subgoals in \( R \). It considers these subsets in increasing order of the sizes of the subsets. For each subset it forms a candidate CQAC query whose body is a conjunction of all the subgoals in all \( V' \)'s in that subset and whose head is identical to the head of \( R \). If any (safe) candidate has an expansion that is equivalent to the expansion of the full MiCR, then algorithm CB-MiCR declares this candidate to be a globally minimal 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 globally minimized MiCR happens to be the same as the full MiCR, that is, \( R' \) is \( R \) itself.

### 4.3.4 Bucket Construction in Pruned-MiCR: Illustrative Example

**EXAMPLE 4.3.**

\[
Q(X, Z) \leftarrow p(X, Y, Y, X, X), s(Z, Z) \ Z < 3.
\]

\[
V_1() \leftarrow p(X_1, A_1, B_1, X_1, C_1).
\]

\[
V_2(X_2) \leftarrow p(X_2, A_2, B_2, X_2, C_2).
\]

\[
V_3(X_3) \leftarrow p(X_3, A_3, B_3, X_3, X_3).
\]

\[
V_4(X_4) \leftarrow p(C_4, A_4, B_4, X_4, X_4).
\]

\[
V_5(B_5) \leftarrow p(A_5, V_5, Y_5, B_5, C_5).
\]

\[
V_6(Z_6) \leftarrow s(Z_6, T_6).
\]

\[
R(X, Z) \leftarrow V_1(), V_2(X), V_3(X), V_4(X), V_5(X), V_6(Z), \ Z < 3.
\]

\[
R'(X, Z) \leftarrow V_3(X), V_5(X), V_6(Z), \ Z < 3.
\]

In Example 4.3 suppose we are trying to find the full MiCR and a globally minimal minimized MiCR of the query \( Q \) using the set of views \( V = \{V_1, V_2, V_3, V_4, V_5, V_6\} \).
Computing the full MiCR

As outlined in Section 4.3.1 (page 71) and in Algorithm 3 (page 70), the algorithm full-MiCR 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 $R(X, Z) :- \emptyset$). The algorithm 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 $V$ whose heads can be added to $R$. In Example 4.3, on examining $V_1$, $V_1()$ gets added to $R$. Similarly, for $V_2$ algorithm pruned-MiCR 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

Algorithm pruned-MiCR examines each relational subgoal that is in the body of each of $V_1()$, $V_2(X)$, $V_3(X)$, $V_4(X)$, $V_5(X)$, and $V_6(Z)$ that are included in $R$ and processes them as follows.

$V_1$: For view $V_1$, for the view subgoal $p(X_1, A_1, B_1, X_1, C_1)$ and the query subgoal $p(X, Y, Y, X, X)$, algorithm pruned-MiCR (Algorithm 4 on page 72) 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 $V_2(X)$ 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 $X$ that is not contributed by $V_1$. Hence, algorithm pruned-MiCR 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.
V4: For view V4, for the view subgoal p(C4, A4, B4, X4, X4) and the query subgoal p(X, Y, Y, X, X), pruned-MiCR does not construct any new bucket since the existing bucket that contains V3(X) already covers the p-subgoal in Q in a tighter way.

V5: For view V5, for the view subgoal p(A5, Y5, Y5, B5, C5) and the query subgoal p(X, Y, Y, X, X), pruned-MiCR constructs a new bucket and inserts V5(X) as an entry in that bucket. The algorithm sees that currently there is only one bucket for the p-subgoal in Q, and V5(X) (that has the same value for the second and third arguments in its p-subgoal) can cover a “requirement” of the full MiCR R that the existing bucket does not cover. Hence a new bucket is created and V5(X) is inserted into it.

V6: For view V6, for the view subgoal s(Z6, T6) and the query subgoal s(Z, Z), pruned-MiCR constructs a new bucket and inserts V6(Z), Z < 3 as an entry in that bucket.

Minimum Set Cover and Finding the Globally Minimal Minimized MiCR

At the end of the bucket-forming procedure, finally there are three buckets and the minimum-set-cover routine returns \{V3(X), V5(X), V6(Z), Z < 3\} as answer, so pruned-MiCR constructs

\[ R'(X, Z) : V3(X), V5(X), V6(Z), Z < 3. \]

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

If algorithm CB-MiCR (Algorithm 5 on page 74) is used in place of algorithm pruned-MiCR (when working on the inputs of Example 4.3) instead of constructing and using buckets, it forms candidates by considering all subsets of the relational subgoals of \( R \). CB-MiCR first considers all subsets of size one, and does not find any candidate 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 \{V3(X), V5(X), V6(Z), Z < 3\}, and determines that the expansion of this candidate is equivalent to the expansion of the
full MiCR. Hence, it returns
\[ R'(X, Z) \leftarrow V_3(X), V_5(X), V_6(Z), Z < 3. \]
as the globally minimal minimized MiCR.

### 4.3.5 Incompleteness of Algorithm Pruned-MiCR

Algorithm pruned-MiCR may not always be able to find a minimized MiCR \( R' \). (But whenever pruned-MiCR does output \( R' \), it is guaranteed that \( R' \) is a globally minimal minimized MiCR equivalent to the full MiCR \( R \); please see Section 4.4 on page 84.) We now show an example where algorithm pruned-MiCR is incomplete.

**EXAMPLE 4.4.** Consider query \( Q \) and the four views \( V_1, V_2, V_3, \) and \( V_4 \) as follows.
\[
Q \leftarrow p(X, X), r(X, X).
V_1(X) \leftarrow p(X, X).
V_2(X) \leftarrow p(X, Y), r(X, Y).
V_3() \leftarrow r(X, X).
V_4() \leftarrow p(X, Y).
R \leftarrow V_1(X), V_2(X), V_3(), V_4().
\]
\( R \) is a MiCR of \( Q \) using \( \{V_1, V_2, V_3, V_4\} \). Notice that \( R \) is not a minimized MiCR (for example, \( V_4 \) is redundant in \( R \)). The (“requirement” of the) equality of the two variables in the \( p \)-subgoal of \( Q \) is covered by \( V_1 \). Similarly, the equality of the two variables in the \( r \)-subgoal of \( Q \) is covered by \( V_3 \), and the equality of the first variable in the \( p \)-subgoal of \( Q \) with the first variable in the \( r \)-subgoal of \( Q \) as well as the equality of the second variable in the \( p \)-subgoal of \( Q \) with the second variable in the \( r \)-subgoal of \( Q \) is covered by \( V_2 \). Pruned-MiCR deems that \( V_1 \) covers the \( p \)-subgoal of \( Q \) more tightly than \( V_2 \), and also that \( V_3 \) covers the \( r \)-subgoal of \( Q \) more tightly than \( V_2 \). Hence, \( V_2 \) is not entered into either bucket. Also, since \( V_1 \) covers the \( p \)-subgoal of \( Q \) more tightly than \( V_4 \), \( V_4 \) is not entered into any bucket.

Thus, the candidate CQAC query generated by pruned-MiCR is
\[ R' \leftarrow V_1(X), V_3(). \]
However pruned-MiCR finds that for this \( R' \), \( (R')^{exp} \nsubseteq R^{exp} \) is not true. Therefore, it does not return any minimized MiCR and returns only the full MiCR \( R \).
More Details and Examples on the Incompleteness of Pruned-MiCR

Algorithm pruned-MiCR (Algorithm 4 on page 72) is incomplete because it constructs and maintains only a subset of the set of all possible buckets. We now describe some design decisions that influenced the types of buckets that pruned-MiCR chose to maintain versus those that it chose to omit. The corresponding experimental evaluation of the choice of buckets is presented in Section 5.2 (page 97).

The process of obtaining a globally minimal minimized MiCR can be thought of as starting with the full MiCR $R$, and then dropping some of the relational subgoals in it to arrive at a minimal MiCR $R'$. Normally, dropping a relational subgoal from a CQAC query makes it a “more containing” CQAC query. But whenever algorithms pruned-MiCR and CB-MiCR output an $R'$, they are careful to drop only “redundant” subgoals in going from $R$ to $R'$. (A subgoal $g$ in a CQAC query $R$ is a redundant subgoal in $R$, if the CQAC query $R''$ obtained by dropping $g$ from $R$, has an expansion that is equivalent to the expansion of $R$.) The following strategies can be used to select the subgoals that are to be retained:

1. Naive strategy: Algorithm CB-MiCR adopts the strategy of exhaustive search, and tries all possible subsets of the set of relational subgoals in the full MiCR. If the full MiCR has $s$ relational subgoals, then in the worst case CB-MiCR has to try all the $2^s$ subsets, performing a CQAC containment check (Section 2.1.2 on page 14) for each subset. In spite of this high complexity, the advantage of algorithm CB-MiCR lies in its ability to always find a globally minimal minimized MiCR on all inputs.

2. Bucket strategy: Algorithm pruned-MiCR adopts this strategy and uses buckets to model the various “requirements” in the full MiCR. Each of the following REQ-1, REQ-2, and REQ-3 is an example of a requirement:

1. **REQ-1:** The full MiCR has in it a subgoal with predicate name $t$.

2. **REQ-2:** The second and fifth arguments in the $p$-subgoal of the query should be equal since they are represented by the same variable $X$ in the full MiCR.

3. **REQ-3:** The first argument of the $s$-subgoal of the query is the same as the third argument of the $p$-subgoal of the query.

If a MiCR algorithm that uses the bucket strategy of pruned-MiCR chooses to construct
a bucket for every possible requirement, then it is guaranteed to be complete (i.e., the CQAC answer that it produces is guaranteed to have an expansion that is equivalent to the expansion of the full MiCR). However, in such a case the number of buckets will be very large, and consequently the time complexity for constructing and populating these buckets will also be very large. So in designing pruned-MiCR we choose to maintain buckets to model only those requirements that are commonly encountered and that we deem as important. Each bucket that is left out by pruned-MiCR represents an “uncovered” requirement, and on some inputs this could result in pruned-MiCR not being able to output the globally minimal minimized MiCR. In designing a bucket-based algorithm, as we increase the number of buckets that we choose to maintain, we decrease the number of uncovered requirements. Hence we find the minimal MiCR on more inputs and gain more completeness at the expense of efficiency.

**The Choice of Buckets in Pruned-MiCR**

The algorithm pruned-MiCR (Algorithm 4 on page 72) is designed to maintain buckets for the following requirements:

- the requirements about the presence of subgoals with specific predicate names (i.e., if the full MiCR has a $t$-subgoal as stated in requirement REQ-1, then algorithm pruned-MiCR makes sure that the expansion of the CQAC query $R'$ that it constructs also has a $t$-subgoal)

- the requirements about equated variables within one subgoal (i.e., since the equated variables in the requirement REQ-2 occur within the same subgoal, pruned-MiCR maintains a bucket for REQ-2, but since the equated variables in the requirement REQ-3 occur across two different subgoals, pruned-MiCR does not maintain a bucket for REQ-3).

Consider Example 4.5 (page 82), Example 4.6 (page 82), and Example 4.7 (page 83), along with Table 4.1 (page 82), Table 4.2 (page 83), and Table 4.3 (page 83), respectively. In Example 4.5, algorithm pruned-MiCR is able to find the minimal MiCR in all cases. In Example 4.6 as well, algorithm pruned-MiCR is able to find the minimal MiCR in all cases except for the following case: Since $V_4$ is more restrictive than $V_3$, a rewriting that uses $V_4$ is a more minimally containing rewriting as compared to a rewriting that uses
V3. But pruned-MiCR cannot detect this difference and treats V3 and V4 as equivalent for its purposes. However, an algorithm that covers (i.e., maintains buckets to represent) the requirement of variables equated across different subgoals of the query would have been able to detect that V4 is more minimally containing than V3. But even such an algorithm that finds the correct minimal MiCR for Example 4.6 still leaves uncovered other requirements such as those in Example 4.7. To ensure that we find the correct minimal MiCR for Example 4.7, we would need an algorithm that not only maintains buckets for equated variables across subgoals (this is needed because in Example 4.7 the third argument of the p-subgoal is the same as the first argument of the s-subgoal), but also maintains buckets for the requirements of multiple variable names (this is needed because in Example 4.7 the whole statement “the first and second arguments of the p-subgoal are equal to each other AND the third argument of the p-subgoal is equal to the first argument of the s-subgoal” is a requirement, say REQ-4, involving multiple variable names that needs to be modeled by a single bucket).

Thus, an algorithm that finds the correct minimal MiCR for Example 4.7 would need to form connected components (of subgoals of the query) as follows: (i) initialize: each subgoal in the query belongs to a separate unconnected component, (ii) procedure: two components should be connected if and only if their subgoals share a variable, (iii) terminate: when no more connections are possible. After forming connected components, the algorithm would need to maintain buckets to model every possibility within each connected component, as illustrated in Example 4.8. The query Q in Example 4.8 has 3 occurrences of the variable X and 3 occurrences of the variable Y all in one connected component. In the worst case, we need a separate bucket to model every combination of two or more X’s and every combination of two or more Y’s and also to model the various combinations of these X’s and Y’s taken together. This leads to a very large number of buckets in all. Such complex algorithms that maintain a large number of buckets, essentially move towards trying to guarantee the ability of finding the minimal MiCR on more and more inputs. But in that case the bucket strategy may no longer prove worthwhile and it may be more practical to directly adopt CB-MiCR. Note that the example inputs illustrated in this section represent some of the tradeoff points between the extremes of pruned-MiCR and CB-MiCR, and intermediate MiCR algorithms (with increasing completeness but decreasing efficiency) can be designed for each one as we progress from Example 4.5 through Example 4.6 and
Table 4.1: The minimal MiCR for the different $V$'s in Example 4.5 (page 82).

<table>
<thead>
<tr>
<th>If $V$ is the set</th>
<th>Then the minimal MiCR is</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_1$</td>
<td>$V_1()$</td>
</tr>
<tr>
<td>$V_2$</td>
<td>$V_2()$</td>
</tr>
<tr>
<td>$V_3$</td>
<td>$V_3()$</td>
</tr>
<tr>
<td>$V_4$</td>
<td>$V_4()$</td>
</tr>
<tr>
<td>$V_5$</td>
<td>$V_5()$</td>
</tr>
<tr>
<td>$V_1, V_2$</td>
<td>$V_2()$</td>
</tr>
<tr>
<td>$V_1, V_2, V_3$</td>
<td>$V_2(), V_3()$</td>
</tr>
<tr>
<td>$V_2, V_3, V_4$</td>
<td>$V_4()$</td>
</tr>
<tr>
<td>$V_4, V_5$</td>
<td>$V_5()$</td>
</tr>
<tr>
<td>$V_2, V_3, V_4, V_5$</td>
<td>$V_5()$</td>
</tr>
<tr>
<td>$V_1, V_2, V_3, V_4, V_5$</td>
<td>$V_5()$</td>
</tr>
</tbody>
</table>

Example 4.7 to Example 4.8

**EXAMPLE 4.5.**

$Q() \ :- p(X, X, Y, Y)$.
$V_1() \ :- p(L, M, N, O)$.
$V_2() \ :- p(L, L, N, O)$.
$V_3() \ :- p(L, M, N, N)$.
$V_4() \ :- p(A, A, B, C), p(D, E, H, H)$.
$V_5() \ :- p(F, F, G, G)$.

**EXAMPLE 4.6.**

$Q() \ :- p(X, X), s(X, X)$.
$V_1() \ :- p(A, A), s(B, C)$.
$V_2() \ :- p(D, E), s(F, F)$.
$V_3() \ :- p(G, G), s(H, H)$.
$V_4() \ :- p(I, I), s(I, I)$.
Table 4.2: The minimal MiCR for the different $V$’s in Example 4.6 (page 82).

<table>
<thead>
<tr>
<th>If $V$ is the set</th>
<th>Then the minimal MiCR is</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_1$</td>
<td>$V_1()$</td>
</tr>
<tr>
<td>$V_2$</td>
<td>$V_2()$</td>
</tr>
<tr>
<td>$V_3$</td>
<td>$V_3()$</td>
</tr>
<tr>
<td>$V_4$</td>
<td>$V_4()$</td>
</tr>
<tr>
<td>$V_1, V_2$</td>
<td>$V_1(), V_2()$</td>
</tr>
<tr>
<td>$V_1, V_2, V_3$</td>
<td>$V_3()$</td>
</tr>
<tr>
<td>$V_1, V_2, V_4$</td>
<td>$V_4()$</td>
</tr>
<tr>
<td>$V_3, V_4$</td>
<td>$V_4()$</td>
</tr>
<tr>
<td>$V_1, V_2, V_3, V_4$</td>
<td>$V_4()$</td>
</tr>
</tbody>
</table>

Table 4.3: The minimal MiCR for the different $V$’s in Example 4.7 (page 83).

<table>
<thead>
<tr>
<th>If $V$ is the set</th>
<th>Then the minimal MiCR is</th>
</tr>
</thead>
<tbody>
<tr>
<td>$V_1$</td>
<td>$V_1()$</td>
</tr>
<tr>
<td>$V_2$</td>
<td>$V_2()$</td>
</tr>
<tr>
<td>$V_1, V_2$</td>
<td>$V_2()$</td>
</tr>
</tbody>
</table>

EXAMPLE 4.7.

$Q() : p(X, X, Y), s(Y)$.  
$V_1() : p(A, A, B), p(C, D, E), s(E)$.  
$V_2() : p(F, F, G), s(G)$.

EXAMPLE 4.8.

$Q() : p(X), s(X), t(X, Y), u(Y), w(Y)$.  
$V_1() : p(A), s(B), t(C, D), u(E), w(F)$.  
$V_2() : p(G), s(G), t(C, D), u(E), w(F)$.  
$V_3() : p(A), s(G), t(G, D), u(E), w(F)$.  
$V_4() : p(G), s(B), t(G, D), u(E), w(F)$.  
$V_5() : p(A), s(B), t(C, H), u(H), w(F)$.  
$V_6() : p(A), s(B), t(C, D), u(H), w(H)$.  
$V_7() : p(A), s(B), t(C, H), u(E), w(H)$.  
$V_8() : p(A), s(G), t(G, H), u(H), w(F)$.  

$\square$
4.3.6 Complexity of Algorithm Pruned-MiCR

We now discuss the complexity of algorithm pruned-MiCR. Let $r$ be the maximum number of relational subgoals in $Q$ and in any view in $V$. Let $p$ be the number of relational subgoals 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 (the one that finds the minimum set cover by simply examining all subsets) takes at most $2^p$ 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 $O(2^p)$. □

4.4 Correctness of the MiCR Algorithms

In this section, we present results on the soundness and completeness of algorithm full-MiCR (Algorithm 3 in Section 4.3.1 on page 71) and algorithm pruned-MiCR (Algorithm 4 in Section 4.3.2 on page 73). The algorithm full-MiCR is sound and complete (Theorem 4.1 on page 84 and Theorem 4.2 on page 85, respectively) for all problem inputs for which the homomorphism property (Definition 3.9 on page 25 in Section 3.2) holds between the expansion of the rewriting and the input query.

**Theorem 4.1. (Soundness Theorem for Full-MiCR)** Given a CQAC query $Q$ and CQAC views $V$. For any CQAC query $R$ (defined in terms of $V$) that is output by algorithm full-MiCR, there exists a single containment mapping from $R^{exp}$ to $Q$ that establishes $Q \sqsubseteq R^{exp}$.
Proof. 1. Let $s$ be a relational subgoal in $R$. Then 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 view $V$ (line 2 in Algorithm 3 on page 70) and then decides to add subgoal $s$ to the body of $R$ (line 10 in Algorithm 3 on page 70), there must exist a containment mapping, call it $\phi$, from the body of $V$ to the body of $Q$ that the algorithm has examined (line 3 in Algorithm 3 on page 70).

Consider mapping $\phi'$ that is the same as mapping $\phi$ except that the domain of mapping $\phi'$ is restricted to be the distinguished variables in $V$. Let CQAC query $\bar{s}$ be the result of applying $\phi'$ to the CQAC query $V$.² Let $ac_s$ be the ACs that are conjoined with $s$ while adding it to the body of $R$ (line 10 in Algorithm 3 on page 70, note that each AC in $ac_s$ is either an AC in $Q$ or is an AC that is implied by the ACs in $Q$). Let $\tilde{s}$ denote the CQAC query obtained from CQAC query $\bar{s}$ by conjoining the extra ACs $ac_s$ to its body. Also consider the mapping $\mu_s$ which is the same as mapping $\phi$ except that each distinguished variable in $V$ that was in the domain of $\phi$ has been replaced by its image under $\phi$.³ Now, $\mu_s$ is a containment mapping from the body of $\tilde{s}$ to the body of $Q$ that establishes $Q \sqsubseteq \tilde{s}$.

2. Consider mapping $\mu_R$ obtained as follows. The domain of $\mu_R$ is the union of the domains of the mappings $\mu_s$ obtained from each $s$ in $R$, and the range of $\mu_R$ is the union of the ranges of the mappings $\mu_s$ obtained from each $s$ in $R$. For any containment mapping $\mu_s$ obtained from an $s$ in $R$, if in $\mu_s$, the variable (or constant) $X$ (in the domain of $\mu_s$) maps to the variable (or constant) $Y$ (in the range of $\mu_s$), then $\mu_R$ is constructed in such a way that in the mapping $\mu_R$ too, $X$ maps to $Y$. (Note that such a construction of $\mu_R$ is possible because each $\tilde{s}$ makes use of fresh names for its nondistinguished variables, and makes use of the names of variables in $Q$ for its distinguished variables.) By definition of containment mapping and by the construction of $R$ in algorithm full-MiCR, $\mu_R$ is a containment mapping from the body of $R^{exp}$ to the body of $Q$, hence $Q \sqsubseteq R^{exp}$.

\[
\square
\]

Theorem 4.2. (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

²That is, $\bar{s}$ is obtained from $V$ by replacing each distinguished variable in $V$ by its image under $\phi$.
³That is, if under $\phi$, the distinguished variable $X$ in $V$ mapped to variable $A$ in $Q$, then in mapping $\mu_s$, variable $A$ now maps to itself.
containment mapping \( \mu_P \) from \( P^{exp} \) to \( Q \) that establishes \( Q \subseteq P^{exp} \). Then (1) the output of algorithm full-MiCR is not empty, and (2) denoting by \( R \) the CQAC output of algorithm full-MiCR we have that \( R^{exp} \subseteq P^{exp} \).

Proof. 1. Let \( s \) be a relational subgoal in \( P \). Let \( \mu_s \) be the mapping that is obtained from \( \mu_P \) by restricting its domain to those subgoals in \( P^{exp} \) that 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 \( V \). Since algorithm full-MiCR examines each view in \( V \) (line 2 in Algorithm 3 on page 70) and each possible containment mapping (line 3 in Algorithm 3 on page 70), it must have examined \( V \) and must have added \( s \) as a relational subgoal in \( R \). This proves (1).

2. The reasoning in item 1 is true for every relational subgoal \( s \) in \( P \), therefore \( R^{exp} \subseteq P^{exp}_0 \), where \( P_0 \) is a conjunction of all the relational subgoals in \( P \). Since \( Q \subseteq P^{exp} \), the ACs in \( P \) are not more restrictive than the ACs in \( Q \), 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 \( \mu_P \) is allowed to take in \( Q \). Also, the extra ACs (i.e., the ACs other than those that are in the body of any \( s \)) that are added to \( R \) (line 10 in Algorithm 3 on page 70) are taken exactly as they are from the ACs in \( \text{closure}(AC(Q)) \). Hence, the values that \( X \) is allowed to take in \( P^{exp} \) cannot be more restrictive than the values of the corresponding variable in \( R^{exp} \) as well. Thus, \( R^{exp} \subseteq P^{exp} \). This proves (2).

\[ \square \]

Theorem 4.3. (Soundness and Completeness (in the MiCR sense) Theorem for Pruned-MiCR) Given a CQAC problem input \((Q, V, R)\), where \( R \) is a CQAC MiCR for CQAC query \( Q \) using CQAC views from the set \( V \), let \( R' \) be the CQAC rewriting that is output by algorithm pruned-MiCR as the minimized MiCR. Then \( R' \) is a globally minimal minimized CQAC MiCR for \( Q \) and \( V \).

Proof. For any CQAC rewriting \( R' \) (defined in terms of \( V \)) that is output as the minimized MiCR by algorithm pruned-MiCR, to prove that \( R' \) is a globally minimal minimized CQAC MiCR for \( Q \) and \( V \), we must show that (a) \( (R')^{exp} \equiv R^{exp} \), and (b) there does not exist any CQAC rewriting \( R'' \) (defined in terms of \( V \)) 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 \( R' \).

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, \( R'^{\exp} \subseteq (R')^{\exp} \). Since \( R' \) is output by algorithm pruned-MiCR, it has been explicitly checked (in line 17 of Algorithm 4 on page 72) that \((R')^{\exp} \subseteq R^{\exp}\) is true. Hence, \((R')^{\exp} \equiv R^{\exp}\) and (a) is proved.

2. The proof of (b) follows from the correctness of the minimum-set-cover algorithm.
   (i) At the end of the bucket-forming procedure (by line 14 in Algorithm 4 on page 72) of algorithm pruned-MiCR, each bucket represents a requirement\(^4\) (or condition) 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 requirement being represented by bucket \( B \) will not hold true 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.
   (ii) The minimum-set-cover algorithm picks not just any set cover, but it picks a 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 \( R'' \) has at least one entry from each bucket (by (i) above, this is required if \( R'' \) has to have \((R'')^{\exp} \equiv R^{\exp}\)), 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 (ii) above, we conclude that such an \( R'' \) does not exist.

   Thus, when algorithm pruned-MiCR outputs a minimized version \( R' \) of the full MiCR \( R \) (line 18 in Algorithm 4 on page 72), then by definition \( R' \) is a globally minimal minimized CQAC MiCR for the given query \( Q \) using the given viewset \( V \).

   While being complete in the sense of Theorem 4.3, algorithm pruned-MiCR is not complete in the sense that it may not output the minimized version \( R' \) at all, that is, it

\(^4\)For example, for a bucket labeled \((p(A, B, B), p(X, Y, Y))\), the requirement is that the last two arguments of the \( p \)-subgoal have the same value. Section 4.3.5 on page 78 has more details on requirements.
may output just $R$ instead of outputting both $R$ and $R'$. (The reason for this behavior is discussed in Section 4.3.5 on page 78). However, algorithm pruned-MiCR is always sound for any given CQAC problem input $(Q, V, R)$.

We conclude our discussion on MiCRs by showing that the idea of algorithm 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 algorithm Build-MaxCR of Section 3.3 (page 26). Suppose that algorithm Build-MaxCR has been executed to obtain the MCR of query $Q$ using the viewset $V$. Then each CQAC component, say $R$, in the UCQAC output of algorithm Build-MaxCR can be minimized by running algorithm pruned-MiCR on input $(Q, V, R)$, and then replacing the CQAC query $R$ in the Build-MaxCR output, by the answer $R'$ output by algorithm pruned-MiCR. 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 MCR 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.

4.5 Variable Types in Arithmetic Comparisons

Checking for containment of two CQAC queries may require constructing all possible total orders of the variables and constants in the canonical databases for the queries; see Chapter 2 (page 10) for the definitions. 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 section, we answer in the positive the question of whether one can reduce the number of total orders that must be checked while checking for containment among CQAC queries. The reduction in our proposed approach 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 after introducing the required terminology.

We define the type of each variable appearing in a CQAC query, as follows. For
each argument of each predicate appearing in the query body, we assign to the argument a distinct equivalence class. We then merge the equivalence classes of those arguments that have a shared variable (see Section 2.1 on page 10 for the definition). If variables \( X \) and \( Y \) appear in the same AC, then we merge the equivalence classes for \( X \) and \( Y \). We refer to the final merged equivalence classes as the types of each argument and of its corresponding variable in the query.

Let \( Q_1 \) and \( Q_2 \) be two CQAC queries. Recall (Section 2.1.2 on page 14) that \( Q_2 \) is contained in \( Q_1 \) iff \( I : \beta_2 \Rightarrow \mu_1(\beta_1) \lor \mu_2(\beta_1) \ldots \), where \( \mu_1, \mu_2, \ldots \) are all the containment mappings from \( Q_1 \) to \( Q_2 \).

The projection \( \text{projection}(I, i) \) of the implication \( I \) on a type \( i \) is obtained by computing the closure of each of the disjuncts in the left-hand side (lhS) and right-hand side (rhS) of the implication and then eliminating from the implication all those ACs which contain variables that are not of type \( i \). We say that an AC is an AC of type \( i \) only if all variables appearing in that AC are of type \( i \).

**Lemma 4.1.** In the implication \( I : \beta_2 \Rightarrow \mu_1(\beta_1) \lor \mu_2(\beta_1) \ldots \), if the rhS of \( I \) has an AC of type \( i \), then each disjunct in the rhS of \( I \) has an AC of type \( i \).

**Proof.** Each disjunct in the rhS of \( I \) is the result of applying some mapping \( \mu_i \) to the ACs of the same query (namely the query \( Q_1 \)). If the rhS of \( I \) contains an AC of type \( i \), then that AC must have come from one of the disjuncts. If one of the disjuncts has an AC of type \( i \), then the body of query \( Q_1 \) has an AC of type \( i \), and hence the image of that AC under any mapping \( \mu_1, \mu_2, \ldots \) must also produce an AC of type \( i \) in the corresponding disjunct in the rhS.

**Theorem 4.4.** If the implication \( I : \beta_2 \Rightarrow \mu_1(\beta_1) \lor \mu_2(\beta_1) \ldots \) holds true, then \( \forall i, \) where \( i \) is a merged equivalence class constructed from \( Q_1 \) and \( Q_2 \), \( \text{projection}(I, i) \) holds.

**Proof.** Consider a specific type \( i \). On the lhS of \( \text{projection}(I, i) \), we have all the ACs in the closure of \( \beta_2 \) that are of type \( i \). On the rhS of projection \( \text{projection}(I, i) \), we have all the ACs in the closure of the disjuncts in the rhS of \( I \) that are of type \( i \). Assume that \( \text{projection}(I, i) \) does not hold. Then there exists a canonical database on which the lhS of \( \text{projection}(I, i) \) holds but on which none of the disjuncts in the rhS of \( \text{projection}(I, i) \) hold. This, along with Lemma 4.1, implies that in each disjunct in the rhS of \( \text{projection}(I, i) \),
there is an AC of type $i$ that does not hold, and that in each disjunct in the rhs of $I$ there is an AC of type $i$ that does not hold. This implies that $I$ does not hold, which is a contradiction.

Theorem 4.4 says that when implication $I$ holds, then its projection implications on each type also hold. However, the inverse of Theorem 4.4 is not true. The following counterexample gives the intuition for why the inverse does not hold.

**EXAMPLE 4.9.** Consider the two queries $Q_1$ and $Q_2$.

$Q_1 \leftarrow p(X,Y), X < 25, Y < 0$.

$Q_2 \leftarrow p(X_1,X_2), p(X_3,X_4), X_1 < 25, X_2 = 0, X_3 = 25, X_4 < 0$.

Consider the implication $I : \beta_2 \Rightarrow \mu_1(\beta_1) \lor \mu_2(\beta_1) \ldots$

That is,

$\left( X_1 < 25, X_2 = 0, X_3 = 25, X_4 < 0 \right) \Rightarrow \mu_1(X < 25, Y < 0) \lor \mu_2(X < 25, Y < 0)$

where, $\mu_1$ is the mapping $\{X \rightarrow X_1, Y \rightarrow X_2\}$ and $\mu_2$ is the mapping $\{X \rightarrow X_3, Y \rightarrow X_4\}$

Hence, the implication $I$ is:

$\left( X_1 < 25, X_2 = 0, X_3 = 25, X_4 < 0 \right) \Rightarrow \left( (X_1 < 25, X_2 < 0) \lor (X_3 < 25, X_4 < 0)\right)$

Clearly, $I$ does not hold because $X_2 = 0$ violates the first disjunct on the rhs and $X_3 = 25$ violates the second one.

Now suppose that variables $X$, $X_1$, and $X_3$ are of type $\text{type}(X)$ and that variables $Y$, $X_2$, and $X_4$ are of type $\text{type}(Y)$. Projecting $I$ on $\text{type}(X)$ produces $\text{projection}(I, \text{type}(X))$:

$\left( X_1 < 25, X_3 = 25 \right) \Rightarrow (X_1 < 25) \lor (X_3 < 25)$

which holds. Similarly, projecting $I$ on $\text{type}(Y)$ produces $\text{projection}(I, \text{type}(Y))$:

$\left( X_2 = 0, X_4 < 0 \right) \Rightarrow (X_2 < 0) \lor (X_4 < 0)$

which holds too.

Example 4.9 illustrates that even if the individual projection implications $\text{projection}(I, i)$, on all types $i$, do hold, it does not necessarily imply that the full implication $I$ also holds. At the same time (as indicated by Corollary 4.1), it turns out that we can still use the idea of projections to reduce the time taken to check containment between two CQAC queries.
**Corollary 4.1.** Let $I$ be the implication that must be checked to check containment of CQAC query $Q_2$ in CQAC query $Q_1$. If for any type $i_0$, $\text{projection}(I, i_0)$ does not hold, then $Q_2$ is not contained in $Q_1$.

**Proof.** The corollary follows from Theorem 4.4. If $I$ is true, then for all types $i$, $\text{projection}(I, i)$ is also true. But we know that one projection, $\text{projection}(I, i_0)$, is not true. Hence, $I$ too is not true. Since $I$ is not true, $Q_2$ is not contained in $Q_1$.

In case the workload contains many queries for which the containment test fails, then Corollary 4.1 can be used to resolve such cases faster. The corollary suggests that instead of performing the implication check on the full implication $I$, we can check whether the implication holds on each type $i$. If we find even a single type $i_0$ for which the implication $\text{projection}(I, i_0)$ does not hold, then even without testing the overall implication $I$ we already know that $I$ does not hold. Thus while testing for CQAC containment we may be able to detect the failure case much faster.

We now turn to a special case for which the inverse of Theorem 4.4 holds true. This is the case in which the homomorphism property (Definition 3.9 on page 25 in Section 3.2) holds between the two given queries $Q_1$ and $Q_2$, and implication $I$ is simply $\beta_2 \Rightarrow \mu(\beta_1)$, where $\mu$ is the single containment mapping from $Q_1$ to $Q_2$.

**Theorem 4.5.** The implication $I : \beta_2 \Rightarrow \mu(\beta_1)$ holds iff $\text{projection}(I, i)$ holds for each type $i$ in the queries whose containment is being checked.

**Proof.** Theorem 4.4 proves that if $I$ holds, then for each type $i$, $\text{projection}(I, i)$ also holds. For the other direction, suppose that projections $\text{projection}(I, i)$ of $I$ for all types $i$ do hold. Hence, the respective rhs-parts of each projection $\text{projection}(I, i)$ also hold. Thus the conjunction of all these rhs-parts also holds. However, this conjunction is exactly the rhs of $I$. Since the rhs of $I$ holds, we conclude that $I$ holds.

Thus, when the homomorphism property holds between $Q_1$ and $Q_2$, we only need to check whether the projection implications $\text{projection}(I, i)$'s hold in order to infer whether the overall implication $I$ holds. Since checking the $\text{projection}(I, i)$'s may be much faster than checking the entire implication $I$, algorithms that use the idea of Theorem 4.5 may be significantly more scalable than their counterparts that use the standard containment check (Section 2.1.2 on page 14). In our work (see, e.g., the pseudocode of algorithm full-MiCR
given as Algorithm 3 on page 70 in Section 4.3.1, where we assume that the homomorphism property holds, we can make use of Theorem 4.5 to check containment.

In general, all the techniques identified in this section can be used to reduce the runtime of algorithms that check CQAC containment. For example, in the MiCR algorithms, the techniques can be applied when containment checks arise in the following steps:

1. While constructing the full MiCR, the algorithm full-MiCR finds the mapping $\mu$ from the view body to the query body, and it can do this by checking for containment of that view in the query (line 3 in Algorithm 3 on page 70).

2. Before producing as output a globally minimal minimized MiCR $R'$, the algorithm pruned-MiCR checks whether the expansion of the minimized CQAC query that it is considering is contained in the expansion of the full MiCR (line 17 in Algorithm 4 on page 72).

3. For each subset of the set of relational subgoals of the full MiCR, the algorithm CB-MiCR forms a corresponding candidate CQAC query and checks whether its expansion is contained in the expansion of the full MiCR that was input to it (line 8 in Algorithm 5 on page 74).

In the contained-rewriting algorithm 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 CQAC queries. However, the techniques described in this section can still be used at the later stage, in which the individual CQAC queries are being added to the UCQAC answer of Build-MaxCR. When Build-MaxCR is trying to add any CQAC query $P'$ to its UCQAC answer $P$, it can use these techniques to check containment between CQAC query $P'$ and each CQAC query $P_i'$ that is already in $P$ (line 7 and line 11 in Algorithm 2 on page 45).
Chapter 5

Experimental Results

In this chapter, we report the results of the experimental evaluation of the algorithms presented in this thesis. Section 5.1 (page 94) describes our experiments with algorithm Build-MaxCR of Chapter 3 (page 18) and Section 5.2 (page 97) describes our results on the performance of the MiCR algorithms of Chapter 4 (page 66).

For our study, we used randomly generated queries of different shapes, such as chain queries\footnote{In a chain query, all the subgoals in the query form a “chain”. Every subgoal is joined with exactly two subgoals — the subgoal that precedes it in the chain and the subgoal that succeeds it in the chain, and there are no other joins between the subgoals. The only exceptions are the first and the last subgoal in the chain, each of which is joined with exactly one subgoal — the first subgoal does not have any subgoal preceding it, and the last subgoal does not have any subgoal succeeding it.} and star queries\footnote{In a star query, all the subgoals in the query form a “star”. There exists a unique subgoal in the query that is joined with every other subgoal, and there are no joins between the other subgoals.} \cite{SMK97}. We used the random query generator implemented in \cite{PH01} 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 runtimes were averaged over twelve executions, after discarding the...
In this section, we discuss our experiments with algorithm Build-MaxCR. Our results show the scalability and runtime of Build-MaxCR for chain and star queries. Figure 5.1 and 5.2 show the runtime of Build-MaxCR for chain queries and views. Figure 5.2 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. However, for higher values of $k$, as the size of the CQAC-rewriting templates becomes larger, the size of $\text{MaxCrVars}$ (the set of variables and constants used to form MaxCR canonical databases as explained in Section 3.3 on page 26) also increases. For $|\text{MaxCrVars}| = n$ 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 that cannot be detected by single-mapping algorithms.
Figure 5.2: Chain queries with up to 1200 views: scalability of Build-MaxCR with k=2.

Figure 5.3: Star queries with 1 to 20 views: Build-MaxCR runtime.
For example, the rewriting of Example 3.5 (page 34 in Section 3.4) can be found by Build-MaxCR even for the low value of \( k = 1 \).

Figures 5.1 and 5.2 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 5.1, 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 that 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 \( M = \emptyset \) (as shown on line 8 of the Build-MaxCR pseudocode in Algorithm 1 on page 28) 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 CQ queries (rather than CQAC queries), or if there were no nondistinguished variables. (If every variable in every view definition is available in the corresponding viewhead, then it becomes readily accessible to any algorithm that is trying to add ACs to a CQ part in order to form a CQAC rewriting that is contained in the query.) However, in practice, it may not be reasonable to expect such special-case conditions in problem inputs. 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 nonhead variables. The presence of shared, nonhead variables gives rise to multiple-mapping contained rewritings. Our experiments show that algorithm Build-MaxCR can find such rewritings, even for low values of \( k \).

Figure 5.3 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 5.4 is an adaptation of Figure 5.3 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 \( k \), algorithm Build-MaxCR scales well with an increase in the number of views.

5.2 Experiments on the MiCR Algorithms

We performed experiments to compare the execution time of algorithm full+pruned-MiCR (that is, we ran the composite algorithm whose first part is the algorithm full-MiCR described in Section 4.3.1 on page 71 and whose second part is the algorithm pruned-MiCR described in Section 4.3.2 on page 73) with the execution time of algorithm full+CB-MiCR (that is, the composite of algorithm full-MiCR and algorithm CB-MiCR described in Section 4.3.3 on page 74), and we report our results in this section. 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 globally minimal minimized MiCR (i.e., a minimal MiCR) to be significantly lower than the number of joins in a full MiCR.

Our experiments demonstrate that algorithm full+pruned-MiCR speeds up rewriting generation, since it eliminates containment checks by doing an early pruning in the
process of generating a minimal MiCR. The implementations of the two MiCR algorithms differ essentially 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 5.5 shows the results for a chain query with ten subgoals. It shows that the execution time of algorithm full+CB-MiCR 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 full+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 algorithm full+pruned-MiCR is evident even at low coverage values of up to 2.

Figure 5.6 shows that algorithm full+pruned-MiCR executes efficiently even for high coverage values. This is in sharp contrast to algorithm full+CB-MiCR, which takes more than 20 seconds even for coverages below 5. The increase in the runtime of algorithm full+pruned-MiCR at higher coverage values is marginal and stems from the increased time
Figure 5.6: Chain queries with coverage range 0 to 8: full+pruned-MiCR and full+CB-MiCR runtimes.

Figure 5.7: Chain queries with up to 1000 views: scalability of algorithm full+pruned-MiCR.
required for forming the MiCR-buckets. In practice, constructing rewritings in algorithm full+pruned-MiCR 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. Algorithm full+CB-MiCR, however, spends a substantial amount of time in the construction of the rewriting, because for each candidate it has to do a CQAC containment test to check if the candidate is contained in the full MiCR. If the full MiCR contains \( m \) subgoals, then algorithm full+CB-MiCR has to perform potentially \( 2^m \) such containment tests, as would be required if the full MiCR itself turns out to be the minimal MiCR. The scalability of algorithm full+pruned-MiCR makes it useful in finding minimal MiCRs for practical cases involving a large number of applicable views. As seen in Figure 5.7 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 5.8 and 5.9 show the runtime of full+CB-MiCR and full+pruned-MiCR on star queries and views. Figure 5.8 considers 1 to 10 views and Figure 5.9 considers up to 500 views. We used a query with one fact table that was joined with 5 dimension tables.
Figure 5.9: Star queries with up to 500 views: scalability of the MiCR algorithms.

for a total of 6 relational subgoals. The views had between 2 and 8 relational subgoals.

Figure 5.10 shows the runtime of algorithm full+pruned-MiCR for chain queries with and without the containment test that 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 from the input viewset \( \mathcal{V} \) is considered, it may generate 0 or more \( h(\mathcal{V}) \)'s. (Please see lines 2-4 in Algorithm 3 on page 70 for details on the generation of \( h(\mathcal{V}) \)'s.) Every time that an \( h(\mathcal{V}) \) is made available: (i) the number of joins in the full 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 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 \( h(\mathcal{V}) \)'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
Figure 5.10: Chain queries: runtime of algorithm full+pruned-MiCR with and without the final containment test.

Figure 5.11: Comparison of the number of joins in the full MiCR and in the minimal MiCR.
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 \( h(V) \) that is generated. Figure 5.11 illustrates these ideas for the simple example of Figure 5.12.
Figure 5.13: Chain queries with 250 views: Algorithm full+pruned-MiCR always finds the full MiCR. In addition, it correctly finds the minimal MiCR on 96% of the input sets.

Figure 5.14: Chain queries with 300 views: Algorithm full+pruned-MiCR always finds the full MiCR. In addition, it correctly finds the minimal MiCR on 93% of the input sets.
MiCR does not exist, and algorithm full+pruned-MiCR correctly determines this (11%)

The 'minimal' MiCR is the same as the 'full' MiCR, and algorithm full+pruned-MiCR returns the correct 'full' MiCR and the correct 'minimal' MiCR (70%)

The 'minimal' MiCR is different from the 'full' MiCR, and algorithm full+pruned-MiCR cannot find it, so it returns just the correct 'full' MiCR (0%)

The 'minimal' MiCR is different from the 'full' MiCR, and algorithm full+pruned-MiCR returns the correct 'full' MiCR and the correct 'minimal' MiCR (19%)

Figure 5.15: Chain queries with 100 views: Algorithm full+pruned-MiCR always finds the full MiCR. In addition, it correctly finds the minimal MiCR on 100% of the input sets.

MiCR does not exist, and algorithm full+pruned-MiCR correctly determines this (9%)

The 'minimal' MiCR is the same as the 'full' MiCR, and algorithm full+pruned-MiCR returns the correct 'full' MiCR and the correct 'minimal' MiCR (60%)

The 'minimal' MiCR is different from the 'full' MiCR, and algorithm full+pruned-MiCR cannot find it, so it returns just the correct 'full' MiCR (1%)

The 'minimal' MiCR is different from the 'full' MiCR, and algorithm full+pruned-MiCR returns the correct 'full' MiCR and the correct 'minimal' MiCR (30%)

Figure 5.16: Chain queries with 150 views: Algorithm full+pruned-MiCR always finds the full MiCR. In addition, it correctly finds the minimal MiCR on 99% of the input sets.
Figure 5.17: Chain queries with 200 views: Algorithm full+pruned-MiCR always finds the full MiCR. In addition, it correctly finds the minimal MiCR on 94% of the input sets.

Recall (Section 4.3.5 on page 78) that algorithm pruned-MiCR does not construct all possible buckets, and hence it may not be able to find the minimized MiCR on all inputs. We performed experiments to validate our claim that the buckets constructed by algorithm full+pruned-MiCR can correctly find the minimized CQAC query (i.e., the minimal MiCR, which is equivalent to the full MiCR) in most cases in practice. As shown in Figure 5.13 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 at least 250 CQAC views. In only 4 of the 100 input sets, running algorithm full+pruned-MiCR produced a minimized CQAC query that was not equivalent to the full MiCR. In such cases algorithm full+pruned-MiCR does not output a minimal MiCR but outputs only 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 algorithm full+pruned-MiCR), 54 cases in which the minimal MiCR was the same as the full MiCR (and algorithm full+pruned-MiCR was able to find it correctly), and
MiCR does not exist, and algorithm full+pruned-MiCR correctly determines this. The 'minimal' MiCR is the same as the 'full' MiCR, and algorithm full+pruned-MiCR returns the correct 'full' MiCR and the correct 'minimal' MiCR. The 'minimal' MiCR is different from the 'full' MiCR, and algorithm full+pruned-MiCR cannot find it, so it returns just the correct 'full' MiCR.

Figure 5.18: Full+pruned-MiCR with chain queries: distribution of the types of result on the 100 input sets.

Figure 5.19: Full+pruned-MiCR consistently finds the correct answer on the full and the minimal MiCR in over 90% of the cases at all times.
39 cases in which the minimal MiCR was different from the full MiCR (and in these cases too algorithm full+pruned-MiCR was able to correctly find the minimal MiCR). Thus the choice of buckets made in designing algorithm full+pruned-MiCR proved to be appropriate for finding the minimal MiCR in the case of 96% of the input sets. Figure 5.14 represents the results of the next round of experiments. From the same collection of over 1200 CQAC chain queries containing between one and ten relational subgoals, we created another 100 sets of inputs. We increased the number of views available to form the rewriting from 250 to 300, so in this round, each input set contained at least 300 views. We found that in only 7 of the 100 input sets, running algorithm full+pruned-MiCR produced a minimized CQAC query that was not equivalent to the full MiCR, and so in these cases, algorithm full+pruned-MiCR returned only the full MiCR. In all the other cases, algorithm full+pruned-MiCR returned the correct answer on the full MiCR as well as on the minimal MiCR.

Similarly, we performed three additional rounds of experiments, in which the 100 input sets each had at least 100 (Figure 5.15), at least 150 (Figure 5.16), and at least 200 (Figure 5.17) views, respectively. In all five rounds of experiments, algorithm full+pruned-MiCR returned the correct answer on the full MiCR as well as on the minimal MiCR on over 90 of the 100 inputs sets. The combined results are represented in Figure 5.18 —
for each of the five rounds, the 100 input sets are divided according to the four types of outcomes. Figure 5.19 shows how algorithm full+pruned-MiCR consistently returns the correct answer on the full MiCR as well as on the minimal MiCR, even after varying the number of input views. Note that, in some cases, it is possible that the full+pruned-MiCR algorithm’s ability to find the correct minimal MiCR is affected by the order in which the input views are placed in the input file. That is, when algorithm full+pruned-MiCR does not output the minimal MiCR on a particular input file, it may still be possible to output the minimal MiCR by rerunning the algorithm on the same input file, after shuffling the order in which the input views are placed in the file. However, in our experiments, for any input file, if there was even a single possible order of arranging the views in that file, such that algorithm full+pruned-MiCR would not be able to output the minimal MiCR for that order, then that input file was classified as one on which algorithm full+pruned-MiCR does not output the minimal MiCR. In spite of this strict approach, algorithm full+pruned-MiCR was able to find the correct answer on the minimal MiCR for 100, 99, 94, 96, and 93 out of 100 cases at the levels of 100, 150, 200, 250, and 300 views, respectively. That is, algorithm full+pruned-MiCR was able to return the correct answer on the minimal MiCR in more than 90% of the cases at all times, and in practice, depending upon the order in which the views are placed in the input file, this number could be even higher.

As the number of available views increases, there is an increase in the coverage, that is, more view subgoals are available to cover any given query subgoal. With this increase in the number of options available to cover any given query subgoal, there is a corresponding increase in the chances of having a minimal MiCR that is different from the full MiCR. And thus there is also an increase in the possibility that the minimized CQAC query produced on running algorithm full+pruned-MiCR is different from the minimal MiCR (for 100 views there are 0 such cases, while for 300 views there are 7 such cases as shown in Figure 5.19). Figure 5.20 shows this increase in the chances of having a minimal MiCR that is different from the full MiCR — there are less than 20 such cases at 100 views, but almost 75 such cases at the level of 300 views. Figure 5.20 also shows that the number of cases in which no MiCR exists decrease from 11 cases at 100 views to 0 cases at 300 views. This is natural, because as the number of views available to form the rewriting is increased, the chances that at least one of those views will contain the query also increase correspondingly. Thus, in our experiments on input files with at least 300 views each, every input file had at least
one applicable view that contained the query. So at the 300-view level, there are no cases in which the MiCR does not exist.

Thus in summary, our experiments have shown (i) that in addition to always finding the full MiCR, algorithm full+pruned-MiCR also correctly finds the minimal MiCR on many of the inputs, (ii) that for chain queries, algorithm full+pruned-MiCR outperforms algorithm full+CB-MiCR significantly due to its early pruning, and (iii) that algorithm full+pruned-MiCR 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.
Chapter 6

Conclusions

In this chapter, we begin by taking a brief look at some directions for future research related to our work (Section 6.1 on page 111). Finally, we give a summary of our contributions that have been described throughout this thesis (Section 6.2 on page 113).

6.1 Some Directions for Future Work

In this section we point out two possible directions for future work. Section 4.3.5 (page 78) has a detailed discussion on the types of input for which algorithm pruned-MiCR is complete, and the conditions under which it is incomplete. One avenue for future research is to develop other algorithms using the strategy of pruned-MiCR but to maintain more buckets (e.g., multiple-subgoal buckets). Recall that each bucket represents some requirement that holds in the full MiCR $R$. In going from the full MiCR $R$ to the minimal MiCR $R'$ we have to be careful not to add any extra requirements (otherwise the rewriting may no longer remain a containing rewriting) and at the same time not to remove any of the existing requirements (otherwise the rewriting — although it will still remain containing — may no longer be minimally containing). As described in Section 4.3.5 (page 78), it is possible to have modified versions of pruned-MiCR, which model more requirements as compared to those modeled in pruned-MiCR. We could go on adding buckets until we have a version which models all requirements, in which case the containment test that forms

$^1$For example, the bucket represented by the query subgoal $p(A, B, B)$ and the view subgoal $p(X, Y, Y)$ models the requirement that the expansion of the full MiCR has a $p$-subgoal that has the same value for its second and third arguments. Page 79 has more examples of requirements.
the last step of pruned-MiCR would no longer be necessary. In algorithm pruned-MiCR (described in Section 4.3.2 on page 73) we have carefully selected the buckets that represent only those requirements that we deem as important. But it is possible to investigate into the efficiency versus completeness tradeoff, and to see how adding more buckets enables the algorithm to find the minimal MiCR in more cases. This would involve experimentally studying the completeness of higher-complexity variants of pruned-MiCR. The other idea would be to start in the opposite direction and to look for optimized ways of navigating the CB-MiCR search space; that is, to find ways to speed up the exhaustive search in CB-MiCR, perhaps by an early detection and killing of the branches that do not contain the answer.

Another problem for future work involves finding MiCRs in a more expressive language. In place of the language of CQAC queries (which is currently the language in which both pruned-MiCR and CB-MiCR find their rewritings), we could allow UCQAC as the language in which to produce MiCRs. One way would be to find all CQAC queries (overlapping rewritings) that have at least some degree of overlap with the input query $Q$ even if they do not wholly contain $Q$, and then to take unions of such CQAC queries to see if the union contains the entire query $Q$. Another possible route could be to use the fact that any CQAC query can be represented as a union of total-order CQAC queries (Section 3.7.1 on page 55 has a discussion on total-order CQAC queries) and to split the input query $Q$ into its total-order components. We conclude our discussion on this direction of future work, by giving a simple illustrative example of applying the technique.

**EXAMPLE 6.1.**

$$Q() \triangleq p(A, B), A \leq B.$$  

$$V_1() \triangleq p(X, Y), X < Y.$$  

$$V_2() \triangleq p(Z, Z).$$

In Example 6.1 there exists no CQAC containing rewriting of $Q$ using $\mathcal{V} = \{V_1, V_2\}$. However, consider the UCQAC rewriting $R$

$$R() \triangleq R_1 \cup R_2$$

where,

$$R_1() \triangleq V_1().$$  

$$R_2() \triangleq V_2().$$
$R$ is a UCQAC containing rewriting of $Q$ using $\mathcal{V}$. Thus, this example shows that even if there exists no CQAC containing rewriting of a CQAC query $Q$ using CQAC views $\mathcal{V}$, there may exist a UCQAC containing rewriting of the same $Q$ using the same views $\mathcal{V}$.

Since any given CQAC query can be expressed as a union of total-order CQAC queries, in this example, we can consider the CQAC query $Q$ to be the union of total-order CQAC queries $Q_1$ and $Q_2$.

\[
Q() \leftarrow Q_1 \cup Q_2.
\]

where,

\[
Q_1() \leftarrow p(A, B), A < B.
\]

\[
Q_2() \leftarrow p(A, B), A = B.
\]

We can find separately the CQAC containing rewriting for each of $Q_1$ and $Q_2$, and then return the union of these CQAC rewritings as the UCQAC containing rewriting of $Q$. Thus, we obtain $R_1$ as the CQAC containing rewriting of $Q_1$, and $R_2$ as the CQAC containing rewriting of $Q_2$, and then return $R = R_1 \cup R_2$ as the UCQAC containing rewriting of $Q$.

### 6.2 Summary of Contributions

In this section, we briefly recall our contributions that have been described in detail in the earlier parts of this thesis. We summarize our work on contained and containing rewritings, and highlight our main experimental results.

In this thesis we presented techniques for finding view-based rewritings of conjunctive queries with arithmetic comparisons. We started by providing a background and motivation for the problem, and by giving real-world examples of where it is applied. We also gave a brief survey of the known results from the literature, about the various language classes such as CQ queries, CQAC queries, UCQ queries, and UCQAC queries, and about query containment and equivalence. We then described the problem of rewriting queries using views, and formally defined the various types of rewritings — such as equivalent, contained, and containing rewritings.

We studied contained rewritings in detail, with a focus on the problem of finding the maximally contained rewriting for the language of CQAC queries. We presented algorithm Build-MaxCR, which to the best of our knowledge, is the first algorithm to find the $k$-bounded UCQAC maximally contained rewriting of arbitrary CQAC queries and views.
We provided a number of detailed definitions and results in order to establish a formal framework and to develop the machinery that is needed to formally describe algorithm Build-MaxCR and to prove its correctness. We gave illustrative examples to explain the working of the Build-MaxCR pseudocode, for the single-mapping as well as the multiple-mapping case. We pointed out several salient features of Build-MaxCR and discussed possible variations of its processing steps. Finally, we gave detailed proofs for the soundness and completeness of Build-MaxCR.

We then examined containing rewritings, and discussed algorithms full+pruned-MiCR and full+CB-MiCR, which find globally minimal minimized MiCRs. These algorithms first find the full MiCR, and then minimize it with respect to the given set of views in order to eliminate some expensive join operations. Through pseudocode and illustrative examples, we described the bucket-forming strategy of algorithm pruned-MiCR, and the tradeoffs that it involves, in comparison to the exhaustive search of algorithm CB-MiCR. We gave proofs of correctness of the MiCR algorithms and explained the technique of partitioning ACs by variable types. This technique can be used to reduce the time required for checking containment of CQAC queries.

We have implemented algorithm Build-MaxCR, algorithm full+pruned-MiCR, as well as algorithm full+CB-MiCR, and performed extensive experiments to study their performance on various types of queries such as star queries and chain queries, obtained by using the query generator from [PH01]. Our experiments show that Build-MaxCR scales well with an increase in the number of views, however, the number of variables and constants that are used in constructing the MaxCR canonical databases imposes a limitation on its performance. Even at low values of $k$, Build-MaxCR finds rewritings that cannot be found using any algorithm that considers only the single-mapping case. Our experiments with algorithm full+pruned-MiCR and chain queries showed that full+pruned-MiCR produces the correct output for the globally minimal minimized MiCR $R'$ on more than 90% of the input sets, at all times. Also, by design, it always produces the correct output for the full MiCR $R$, on all input sets. Moreover, algorithm full+pruned-MiCR outperforms algorithm full+CB-MiCR significantly due to its early pruning. We also found that algorithm full+pruned-MiCR scales gracefully with an increase in the number of views. Finally, we concluded the thesis with a summary of our contributions and proposed some directions for future research in this area.
Bibliography


