Available online at www.sciencedirect.com
ScienceDirect
Theoretical Computer Science 368 (2006) 88–123
Theoretical Computer Science
www.elsevier.com/locate/tcs

Rewriting queries using views in the presence of arithmetic comparisons☆

Foto Afrati[a,*], Chen Li[b], Prasenjit Mitra[c]

[a] *Electrical and Computing Engineering, National Technical University of Athens, 157 73 Athens, Greece*
[b] *Department of Computer Science, University of California, Irvine, CA 92697-3435, USA*
[c] *College of Information Sciences and Technology, The Pennsylvania State University, University Park, PA 16802, USA*

Received 5 January 2005; received in revised form 22 August 2006; accepted 29 August 2006

Communicated by D. Sannella

Abstract

We consider the problem of answering queries using views, where queries and views are conjunctive queries with arithmetic comparisons over dense orders. Previous work only considered limited variants of this problem, without giving a complete solution. We first show that obtaining equivalent rewritings for conjunctive queries with arithmetic comparisons is decidable. Then, we consider the problem of finding maximally contained rewritings (MCRs) where the decidability proof does not carry over. We investigate two special cases of this problem where the query uses only semi-interval comparisons. In both cases decidability of finding MCRs depends on the query containment test. First, we address the case where the homomorphism property holds in testing query containment. In this case decidability is easy to prove but developing an efficient algorithm is not trivial. We develop such an algorithm and prove that it is sound and complete. This algorithm applies in many cases where the query uses only left (or right) semi-interval comparisons. Then, we develop a new query containment test for the case where the containing query uses both left and right semi-interval comparisons but with only one left (or right) semi-interval subgoal. Based on this test, we show how to produce an MCR which is a Datalog query with arithmetic comparisons. The containment test that we develop obtains a result of independent interest. It finds another special case where query containment in the presence of arithmetic comparisons can be tested in nondeterministic polynomial time.

Keywords: Databases; Query rewriting; Query languages

1. Introduction

In many data-management applications, such as information integration [7,14,22,23,28,37], data warehousing [35], web-site designs [19], and query optimization [12], the problem of answering queries using views [27] is of special significance. The problem is as follows: given a query on a database schema and a set of views over the same schema,

☆ Part of this article was published in Afrati et al. (2002). In addition to the prior materials, this article contains more results (Sections 4 and 5.2 are new) and complete proofs that were not included in the original paper.

* Corresponding author. Tel.: +302102232097; fax: +302107722499.
E-mail addresses: afrati@cs.ece.ntua.gr (F. Afrati), chenli@ics.uci.edu (C. Li), pmitra@ist.psu.edu (P. Mitra).

0304-3975/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2006.08.020

can we answer the query using only the views? To answer the query using the answers to the views efficiently, we rewrite the query using only the view literals. See [26] for a good survey.

A lot of works on query rewriting using views have addressed the problem when both queries and views are conjunctive. In most commercial scenarios, however, users require the flexibility to pose queries using conjunctive queries along with arithmetic comparisons (e.g., $<$, $\leqslant$, $\neq$) between attributes and constants that can take any value from a dense domain (e.g., real numbers). For instance, queries could have conditions such as `carPrice < $3000` and `carYear > 1998`. Similarly, views are also described using conjunctive queries with arithmetic comparisons (CQACs). Thus, the problem of answering queries using views when queries and views have arithmetic comparisons is important in these applications.

Abiteboul and Duschka [1] and Levy et al. [27] have observed that the problem of answering queries using views is closely related to the problem of query containment. Although prior research [24,20] has addressed the issue of containment of CQACs, not many results are known on the problem of query answering and especially query rewriting in the presence of arithmetic comparisons. Abiteboul and Duschka [1] have also shown that the problem is intractable (co-NP hard for data complexity) in many cases.

In this paper, we study the following problem: how can we rewrite a query using views when the query and views are conjunctive with comparisons (e.g., $<$, $\leqslant$, $>$, $\geqslant$, $\neq$)? We take the open-world assumption about the views [17]. That is, the views do not guarantee to export all tuples that satisfy their definitions. Instead, views export only a subset of such tuples. We focus primarily on finding maximally-contained rewritings (MCRs), but we also develop some results on finding equivalent rewritings. Our results on MCRs concern two questions: (1) Given a query and a set of views which are CQACs, is there an MCR in a given query language? (2) If the answer in (1) is positive—and since it is known that the problem of finding an MCR is far beyond PTIME—is there an algorithm can find an MCR efficiently? The following is the structure of the paper and the contributions of this work:

- In Section 2, we review preliminary results in the literature on the problem of rewriting queries using views in the presence of arithmetic comparisons and on query containment, which is recognized to be closely related. We formulate the problem being investigated and discuss its challenges while providing examples. We present also some new observations concerning subcases where the query containment test can be simplified.
- In Section 3, first, we show that the following problem is decidable: for a query and views that are conjunctive with comparisons, is there any equivalent rewriting in the language of unions of CQACs? Then, we turn our attention to MCRs and take question 1 above. In particular, we ask the following decidability question: for a query and views that are conjunctive with comparisons, is there an MCR in the language of unions of conjunctive queries with comparisons? We answer this question positively for two cases: (a) the case where all variables in each view definition also occur in the head and (b) when the homomorphism property holds (i.e., when one mapping suffices to show containment). In fact, we prove that there always exists an MCR in these two cases, and our proof gives an algorithm to find it. An independent contribution in this section (which we need as a tool to prove the results about the existence of an MCR) is the introduction of the notions of AC-containment between two rewritings and of an AC-MCR. We show that we are only interested in AC-MCRs because they produce exactly the same set of answers produced by any MCR.
- In Sections 4 and 5, we take question 2. We develop an efficient algorithm to generate a MCR in (identified sub-cases of the) case where the query has left-semi-interval (LSI) or right-semi-interval (RSI) comparisons, and the views have general arithmetic comparisons, thus answering question 2 for these cases. (In fact, according to [5], these are cases where the homomorphism property holds.) Our algorithm extends the shared-variable-bucket algorithm and similar techniques [30,31] to capture comparisons in an efficient way and finds an MCR in the language of union of CQACs. The proof of soundness and completeness of the algorithm is nontrivial because the algorithm prunes the space of contained rewritings that are considered for candidates to form an MCR significantly. Thus, the challenge is to prove that it does not miss any rewritings that are contained in the query and are in an MCR. In particular, in Section 4, we describe the algorithm and its proof for the conjunctive query (CQ) case, hence our contribution here is providing the proof for soundness and completeness of the algorithm (the algorithm itself is known in the literature, see Table 1). In Section 5, we develop a new efficient algorithm for finding an MCR when the homomorphism property holds and prove its soundness and completeness.
- In Section 6, we answer question 1 for a more general case than queries with only LSI or only RSI comparisons. We study the problem of finding an MCR for queries with semi-interval arithmetic comparisons. We consider a subcase where Datalog programs with semi-interval comparisons are sufficient to express an MCR. We first show

Table 1
Work on finding maximally contained rewritings ("MCR")

Query	Views	MCRs	References
CQ	CQ	Unions of CQs	[21,28,31,30]
Datalog	CQ	Datalog	[18]
Datalog	Union of CQ	Datalog	[3]
CQ with LSI, RSI	CQ with LSI, RSI	Unions of CQs with LSI, RSI	[31, Section 3.2 and 5]
CQ($\neq$)	CQ	co-NP-hard (data complexity)	[1]
CQ with comparisons	CQ with comparisons all variables distinguished	Unions of CQs with comparisons	Section 3.2
CQ with LSI, RSI	CQ with comparisons	Unions of CQs with LSI, RSI	Sections 3.2 and 5
CQ with LSI1, RSI1	CQ with SI	Datalog with SI	Section 6

"CQ" represents "conjunctive queries." For definitions of SI, LSI and RSI see Section 2.1.

that the language of CQACs is not sufficient to express an MCR. Then, we show that query containment in this case can be polynomially reduced to the containment of a conjunctive query in a Datalog query. Based on this result, we develop an algorithm for finding an MCR in the language of Datalog with arithmetic comparisons. For this special case, we also obtain a result of independent interest, i.e., we identify a new class of conjunctive queries with comparisons for which the containment problem is in NP.

1.1. MCR: related work and our contributions

A lot of work has been done on MCRs when queries and views are conjunctive. Specifically efficient algorithms have been discovered and implemented and are known as the bucket algorithms [28,30,31]. The algorithms in [31,30], called, respectively, the MiniCon algorithm and the shared-variable-bucket algorithm are complete for conjunctive queries and views. The algorithm in [31] also handles restricted cases when arithmetic comparisons are present in the views but it is not complete for these cases. Certain answers and their relation to MCRs has been studied in [1,21]. In [1] it has been also proven that MCRs in a polynomially computable language is unlikely to exist in the case the query has inequalities ($\neq$); in particular, it was proven that the data complexity of computing certain answers is co-NP hard. However, recursion in the query does not present a problem when views are conjunctive queries, since in [18] an algorithm is given that computes an MCR of a Datalog query which is a Datalog query itself. However, it has been observed that when views are unions of conjunctive queries then only in special cases we can find an MCR which is a Datalog query [3]. Table 1 summarizes results on the problem of finding MCRs, including those presented in this paper.

In addition, Beeri et al. [8] and Calvanese et al. [9] study the problem of answering conjunctive queries over description logics using views expressed in description logics. Description logics are more expressive than conjunctive queries with comparisons. Also, recent work [2] has developed an efficient algorithm for finding equivalent rewritings in the presence of arithmetic comparisons.

2. Basic definitions

In this section, we give the notation used in the paper, review the problem of query rewriting using views, summarize results in the literature on the containment of CQACs.

2.1. CQACs

We focus on conjunctive queries and views with arithmetic comparisons of the following form:

$$h(\bar{X}) \text{:-}\ g_1(\bar{X}_1), \ldots, g_n(\bar{X}_n), C_1, \ldots, C_m.$$

The head $h(\bar{X})$ represents the results of the query. The body has a set of *ordinary subgoals* $g_1(\bar{X}_1), \ldots, g_n(\bar{X}_n)$, also known as "regular subgoals" or "uninterpreted subgoals" or "ordinary subgoals." Each subgoal $g_i(\bar{X}_i)$ includes a

relation g_i, and a tuple of arguments $\bar{X}_i$ corresponding to the relational schema. An argument can be either a variable or a constant. The variables $\bar{X}$ are called *distinguished* variables. Each C_i is an arithmetic comparison in the form of "$A_1\ \theta\ A_2$," where A_1 and A_2 are variables or constants. If they are variables, they appear in the ordinary subgoals. The operator "θ" is $\neq$, $<$, $\leqslant$, $=$, $>$, or $\geqslant$. We use the terms "inequality" and "arithmetic comparison" or simply "comparison" interchangeably to denote either of the above operators. In addition, we make the following assumptions about the arithmetic comparisons:

1. Values for the arguments in the arithmetic comparisons are chosen from an infinite, totally densely ordered set, such as the rationals or reals.
2. The arithmetic comparisons are not contradictory; that is, there exists an instantiation of the variables such that all the arithmetic comparisons are true.
3. All the comparisons are safe, i.e., each variable in the comparisons appears in some ordinary subgoal.

We use the term *closure*(S) of a set of arithmetic comparisons S, to represent the set of all possible arithmetic comparisons that can be logically derived from S. For example, for the set of arithmetic comparisons $S = \{X \leqslant Y, Y \leqslant 5, Y < Z\}$, the $closure(S) = \{X \leqslant Y, Y \leqslant 5, Y < Z, X < Z, X \leqslant 5\}$. For the sake of simplicity, we use "CQ" to represent "conjunctive query," "AC" for "arithmetic comparison," and "CQAC" for "conjunctive query with arithmetic comparisons." If a CQAC is written as "$Q = Q_0 + \beta$," it means that "β" is the comparisons of Q, and "Q_0" is the query obtained by deleting the comparisons from Q; we refer to Q_0 as the *core* of Q. We say an arithmetic comparison is *open* if its operator is $<$ or $>$; it is *closed* if its operator is $\leqslant$ or $\geqslant$. A query is called *left semi-interval* ("LSI"), if all its comparisons are LSI comparisons, i.e., of the form $X < c$ or $X \leqslant c$, where X is a variable, and c is a constant. A right semi-interval CQAC ("RSI query") and a RSI comparison are defined similarly, i.e., comparisons are of the form $X > c$ or $X \geqslant c$, where X is a variable, and c is a constant. We use the notation semi-interval (SI) to refer to queries and sets of comparisons that contain both LSI and RSI comparisons.

Given a CQ query Q we obtain a *canonical database* D of Q by freezing the variables of Q to constants and then we consider D to contain exactly all the frozen subgoals in the body of the query.

2.2. Query containment and equivalence

The problem of answering queries using views is closely related to the problem of testing for query containment.

Definition 2.1 (*query containment*). A query Q_1 is *contained* in a query Q_2, denoted $Q_1 \sqsubseteq Q_2$, if for any database D, the set of answers of Q_1 on D is a subset of the answers of Q_2 on D. The two queries are *equivalent*, denoted $Q_1 \equiv Q_2$, if $Q_1 \sqsubseteq Q_2$ and $Q_2 \sqsubseteq Q_1$.

Given two conjunctive queries Q_1 and Q_2, $Q_1 \sqsubseteq Q_2$ if and only if there is a *containment mapping* from Q_2 to Q_1, such that the mapping maps a constant to the same constant, and maps a variable to either a variable or a constant. Under this mapping, the head of Q_2 becomes the head of Q_1, and each subgoal of Q_2 becomes *some* subgoal in Q_1 [11].

Let Q_1 and Q_2 be two CQACs. Often we need to test whether $Q_2 \sqsubseteq Q_1$. To do the testing, we can first *normalize* both queries Q_1 and Q_2 to Q'_1 and Q'_2, respectively, as follows:

- For each occurrence of a shared variable X in the normal subgoals except the first occurrence, replace the occurrence of X by a new distinct variable X_i, and add $X = X_i$ to the comparisons of the query; and
- For each constant c in the query, replace the constant by a new distinct variable Z, and add $Z = c$ to the comparisons of the query.

The following theorem is from [20,24,40].

Theorem 2.1. *Let Q_1, Q_2 be CQACs and $Q'_1 = Q'_{10} + \beta'_1$, $Q'_2 = Q'_{20} + \beta'_2$ be the queries after normalization. Let $\mu_1, \ldots, \mu_k$ be all the mappings (homomorphisms) from Q'_{10} to Q'_{20}. Then $Q_2 \sqsubseteq Q_1$ if and only if the following logical implication ϕ is true*:

$$\phi : \beta'_2 \Rightarrow \mu_1(\beta'_1) \vee \cdots \vee \mu_k(\beta'_1).$$

That is, the comparisons in the normalized query Q'_2 logically imply (denoted "$\Rightarrow$") the disjunction of the images of the comparisons of the normalized query Q'_1 under these mappings.

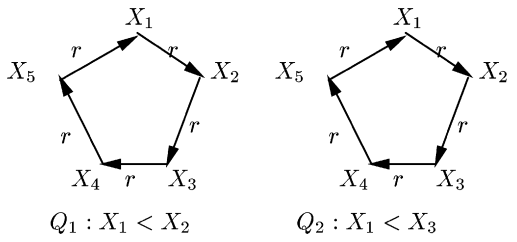

Fig. 1. Graph representations of two equivalent queries.

We refer to ϕ as the *containment entailment*. Notice that in the theorem, the "OR" operation "$\vee$" in the implication is critical, since there might not be a single mapping μ_i from $Q_{1,0}$ to $Q_{2,0}$, such that $\beta_2 \Rightarrow \mu_i(\beta_1)$. The following example shows that to prove containment we need to consider all mappings.

Example 2.1. Consider the following two queries, which are graphically illustrated in Fig. 1:

$$Q_1()\text{:-}\ r(X_1, X_2), r(X_2, X_3), r(X_3, X_4), r(X_4, X_5), r(X_5, X_1), X_1 < X_2.$$
$$Q_2()\text{:-}\ r(X_1, X_2), r(X_2, X_3), r(X_3, X_4), r(X_4, X_5), r(X_5, X_1), X_1 < X_3.$$

Although the two queries have different comparisons, surprisingly, $Q_1 \equiv Q_2$. To show $Q_1 \sqsubseteq Q_2$, we consider the five mappings from the five ordinary subgoals of Q_2 to the five of Q_1. Each mapping corresponds to a "rotation" of the variables. Under these mappings, β_2 becomes $X_1 < X_3$, $X_2 < X_4$, $X_3 < X_5$, $X_4 < X_1$, and $X_5 < X_2$, respectively. We can show that (it is easy to see that if the right-hand side of the implication that follows is false then $X_1 = X_2$):

$$(X_1 < X_2) \Rightarrow (X_1 < X_3) \vee (X_2 < X_4) \vee (X_3 < X_5) \vee (X_4 < X_1) \vee (X_5 < X_2).$$

Therefore, $Q_1 \sqsubseteq Q_2$. Similarly we can prove $Q_2 \sqsubseteq Q_1$. Notice there is no single containment mapping μ_i such that $\beta_2 \Rightarrow \mu_i(\beta_1)$.

Notice that in Example 2.1 we did not need normalization. The following example shows that the containment test of Theorem 2.1 does not go through without having *both* queries normalized before we find the mappings and check the logical implication. Thus, normalization is important and we show below the intuition of this importance.

Example 2.2. Consider the following two queries:

$$Q_1()\text{:-}\ p(A, 4), A < 4.$$
$$Q_2()\text{:-}\ p(X, 4), p(Y, X), X \leqslant 4, Y < 4.$$

Q_2 is contained in Q_1. The informal justification is that if variable X in Q_2 is less than 4 then subgoal $p(A, 4)$ can be mapped to subgoal $p(X, 4)$ and if $X = 4$ then the second subgoal becomes $p(Y, 4)$ and in this case subgoal $p(A, 4)$ maps to $p(Y, 4)$. However, there is only one containment mapping from the ordinary subgoals of Q_1 to Q_2 and if try to work out the logical entailment using this containment mapping, then we will conclude that the logical entailment is false. The normalized versions of the two queries are

$$Q_1'()\text{:-}\ p(A, B), A < 4, B = 4.$$
$$Q_2'()\text{:-}\ p(X, Z), p(Y, X_1), X \leqslant 4, Y < 4, X = X_1, Z = 4.$$

To convince ourselves that normalization of only Q_2 does not suffice, we may want to try to work the test of Theorem 2.1 on Q_1 and Q_2'. The informal reason for why it does not work is that if we consider more than one mapping, then we must map subgoal $p(A, 4)$ to $p(Y, X_1)$ but constant 4 must to map to the same constant 4 and X_1 is not a constant. However, when we deal with Q_1', we do not have this problem because now we map variable B to a variable X_1, which is allowed. Thus, by taking the two mappings on the normalized queries, we have to check the following entailment:

$$X \leqslant 4 \wedge Y < 4 \wedge X = X_1 \wedge Z = 4 \Rightarrow (X < 4 \wedge Z = 4) \vee (Y < 4 \wedge X_1 = 4).$$

If we rewrite the above entailment equivalently we have the (obviously true) entailment:

$$X \leqslant 4 \wedge Y < 4 \wedge X = X_1 \wedge Z = 4 \Rightarrow (X < 4 \vee Y < 4) \wedge (X < 4 \vee X_1 = 4) \wedge (Z = 4 \vee Y < 4) \wedge (Z = 4 \vee X_1 = 4).$$

Another containment test [24,29] is based on canonical databases and does not need normalization. For a CQAC query Q the set of its *canonical databases* with respect to another CQAC query Q' is constructed as follows: we consider the set of the variables of Q and the constants of Q and Q', and we partition this set into blocks with the restriction that two distinct constants do not belong to the same block. For each total ordering of the blocks we construct a canonical database of Q by (a) equating the variables in the same block to a distinct constant (or the constant in the block if there is one) so that the total ordering is satisfied and (b) adding to the canonical database exactly those tuples that result from the frozen relational subgoals of the query.

The test is the following: to test whether $Q_2 \sqsubseteq Q_1$ consider all canonical databases of Q_2 with respect to Q_1. Then $Q_2 \sqsubseteq Q_1$, iff, the following holds on any canonical database D of Q_2: if the head of Q_2 is computed on D then the same head of Q_1 is also computed on D.

2.2.1. Simpler containment tests

In this subsection, we present some observations on special cases where the containment test can be simplified.

There are special cases where the test for containment is simpler, because a single containment mapping suffices for the containment test. We identify in Lemmata 2.1 and 2.2 two such cases, both having special conditions on the queries. Further, in Theorem 2.2, we identify a case where normalization is not necessary.

Lemma 2.1. *Let $Q_1 = Q_{1,0} + \beta_1$ and $Q_2 = Q_{2,0} + \beta_2$ be two CQAC queries. If β_2 is a total ordering of all the variables in $Q_{2,0}$ and all the constants in both Q_1, Q_2, then $Q_2 \sqsubseteq Q_1$ if and only if there is a single containment mapping μ from $Q_{1,0}$ to $Q_{2,0}$, such that $\beta_2 \Rightarrow \mu(\beta_1)$.*

Proof. In every canonical database of Q_2, its variables map to constants that preserve the total order of β_2. Hence, a containment mapping from the variables of Q_1 to a canonical database can be thought of as a mapping μ from the variables of Q_1 to the variables of Q_2 such that $\beta_2 \Rightarrow \mu(\beta_1)$. □

Another case is where queries have comparisons that are LSI or RSI. However, there are subtle subcases that require more than one mapping for the containment test. For a complete analysis on this case, see [5]. The following lemma from [5] presents a simple such case.

Lemma 2.2. *Let $Q_1 = Q_{1,0} + \beta_1$ and $Q_2 = Q_{2,0} + \beta_2$ be two LSI (or RSI) queries. If β_2 does not contain a closed arithmetic comparison when β_1 contains an open arithmetic comparison, then $Q_2 \sqsubseteq Q_1$ if and only if there is a single containment mapping μ from $Q_{1,0}$ to $Q_{2,0}$, such that $\beta_2 \Rightarrow \mu(\beta_1)$.*

Finally there are cases where we do not need to normalize as the following theorem shows.

Theorem 2.2. *Consider two CQAC queries $Q_1 = Q_{1,0} + \beta_1$ and $Q_2 = Q_{2,0} + \beta_2$ that may not be normalized. Suppose β_1 contains only $\leqslant$ and $\geqslant$, and each of β_1 and β_2 does not imply "=" restrictions. Then $Q_2 \sqsubseteq Q_1$ if and only if:*

$$\phi' : \beta_2 \Rightarrow \gamma_1(\beta_1) \vee \cdots \vee \gamma_l(\beta_1),$$

where $\gamma_1, \ldots, \gamma_l$ are all the containment mappings from $Q_{1,0}$ to $Q_{2,0}$.

Proof. The proof is based on the following observation. For all orderings of the variables in Q_2 we consider the set of all those canonical databases of Q_2 such that distinct variables are frozen to distinct constants (also distinct from the constants in the queries). We call them *leading canonical databases*. It is useful to think how we construct a leading canonical database: we consider partitions into blocks (recall how we construct any canonical database) but each block contains only one variable or constant. Thus, leading canonical databases are constructed from the same blocks and differ from each other only on the order of the blocks. Also the following hold: for every canonical database D on

which the head of Q_2 is computed, there is a leading canonical database D' such that (i) there is a homomorphism from the tuples of D' to the tuples of D which preserves comparisons $\leqslant$ and $\geqslant$ and (ii) Q_2 computes its head on D iff Q_2 computes its head on D'. We call D' a *leader* of D.

We give the construction of D' from D. When we construct D we consider certain total ordering among the blocks. Moreover, since the head of Q_2 is computed on D, this total order satisfies the comparisons in Q_2. Observe that all total orderings which satisfy the comparisons (from Q_2) are produced as follows: we partition the variables of Q_2 into blocks and then we define a total order on the blocks. For each block consider the comparisons that are satisfied by instantiating their both variables/constants to elements in this block. Obviously such comparisons are satisfied as to their $=$ option (since we only have $\leqslant$ and $\geqslant$ comparisons and equalities are not implied). Thus, any such comparison can also be satisfied by its instantiation being to variables/constants that are related by $<$ or $>$ instead of $=$. Since the comparisons in the body of the query Q_2 do not have contradictions, there is at least one instantiation of all the variables in the block to distinct constants which satisfy the comparisons in Q_2. We use the order implied by this instantiation for each block to construct the leading canonical database D' which is a leader of D.

The "if" direction: suppose the entailment ϕ' holds. Let D be a canonical database of Q_2 on which its head is computed. According to the above observations, it suffices to consider the leader D' of D and prove that the head of Q_1 is computed on D'. The left-hand side of ϕ' holds on D', hence one of the disjuncts must hold. This implies that there is a homomorphism (the corresponding to the γ of this disjunct) from the relational subgoals of Q_1 to D' which also satisfies the comparisons of Q_1, hence the head of Q_1 is also computed on D'.

The "only if" direction: suppose $Q_2 \sqsubseteq Q_1$. Towards contradiction, suppose ϕ' is false. Then there is a canonical database of Q_2 and hence (according to the discussion above) a leading canonical database D' of Q_2 on which its head is computed and where all disjuncts in ϕ' are false. However, the mappings γ considered in ϕ' are all the mappings that exist from the relational subgoals of Q_1 to D'. Hence, the head of Q_1 is not computed on D' hence $Q_2 \sqsubseteq Q_1$ is false, contradiction. □

2.2.2. Work on complexity of query containment

Chandra and Merlin [11] have shown that the problems of containment, minimization, and equivalence of conjunctive queries are NP-complete. Klug [24] has shown that containment for CQACs is in Π_2^P, whereas when only LSI or RSI comparisons are used, the containment problem is in NP. A containment test based on canonical databases was developed in [24,29]. A more efficient containment test was presented in [20] but the problem still remained in Π_2^P. In [38,39], containment for conjunctive queries with inequality arithmetic comparisons is proven to be Π_2^P-complete. Klug [24] stated that the searching for other classes of CQACs for which containment is in NP is an open problem. We have shown in [5] more classes of CQACs that are in NP. In this paper, we present (in Theorem 6.2) a new class of conjunctive queries with comparisons where containment is in NP.

In [32,15] special cases were identified where conjunctive-query containment is in PTIME. The property that makes it polynomial is acyclicity [32] and its extension, which is defined as bounded query width [15]. Saraiya in [34] proved another case where the containment of conjunctive queries is in PTIME. It is the case where each predicate appears at most twice in the contained query. Kolaitis et al. [25] have studied the computational complexity of the query-containment problem of queries with disequations ($\neq$). In particular, they have shown that the problem remains Π_2^P-hard even in the cases where the acyclicity property holds and each predicate occurs at most three times. However, they proved that if each predicate occurs at most twice then the problem is in coNP.

Containment of a conjunctive query in a Datalog query is shown to be EXPTIME-complete [16,10,33]. Containment of a Datalog query in a conjunctive query is proven to be doubly exponential [13].

Table 2 summarizes work on query containment including our contribution in this paper.

2.3. Rewriting queries using views

The problem of rewriting queries using views [27] is as follows: given a query on a database schema and views over the same schema, can we answer the query using only the answers to the views via a rewriting? The following notations define the problem formally.

Definition 2.2 (*expansion*). The *expansion* of a query P using views V only, denoted by P^{exp}, is obtained from P by replacing all the views in P with their corresponding base relations and comparisons from their definitions. Nondistinguished variables in a view are replaced with fresh variables in P^{exp}.

Table 2
Complexity of query containment: checks whether Q_2 is contained in Q_1

Q_1	Q_2	Complexity	References
CQAC	CQAC	Π_2^P complete	[24,20,39,40]
CQ≠acyclic	CQ≠, each predicate at most 3 times	Π_2^P complete	[25]
CQ≠	CQ≠, each predicate at most twice	coNP	[25]
CQAC homomorphism prop.	CQAC	NP	[24,5]
CQSI1	CQSI	NP	Section 6, Theorem 6.2
CQ	CQ	NP-complete	[11]
CQ	CQ each predicate at most twice	PTIME	[34]
CQ acyclic bounded query width	CQ	PTIME	[32,15]
Recursive Datalog	Nonrecursive Datalog	EXPTIME-complete	[16,10,33]
Nonrecursive Datalog	Recursive Datalog	Doubly exponential	[13]

"CQ" represents "conjunctive queries,"CQ≠" represents "conjunctive queries with only ≠", "CQAC" represents "conjunctive queries with any arithmetic comparisons". For more on notation see definitions in this section.

Definition 2.3 (*rewritings*). Given a query Q and a view set V, a query P is a *contained rewriting* of query Q using V if P uses only the views in V, and $P^{\text{exp}} \sqsubseteq Q$. That is, P computes a partial answer to the query. Given a rewriting language $\mathcal{L}$ (e.g., unions of conjunctive queries with comparisons), we call P an *equivalent rewriting* of Q using V w.r.t. $\mathcal{L}$ if P is in $\mathcal{L}$, and $P^{\text{exp}} \equiv Q$. We call P a MCR of Q using V w.r.t. $\mathcal{L}$ if (1) P is a contained rewriting in $\mathcal{L}$ of Q, and (2) there is no contained rewriting P_1 in $\mathcal{L}$ of Q such that P_1 properly contains P.

Intuitively, an MCR of Q using V w.r.t. a language $\mathcal{L}$ is a query in the language $\mathcal{L}$ that uses only the views. Moreover, the MCR is a contained rewriting, and it computes the maximal answer to Q using the views. In the rest of the paper, unless specified otherwise, we use "rewritings" to mean "contained rewritings."

When the queries and views are expressed as conjunctive queries (without arithmetic comparisons), we know how to find equivalent rewritings (if they exist) and MCRs that are unions of conjunctive queries [26]. However, arithmetic comparisons introduce many complications to the problem. The following examples show some of the subtleties that arise in the presence of arithmetic comparisons.

Example 2.3. This example shows that the comparisons in a rewriting may look very "different" from those in the query and views. Consider the query Q_1 in Example 2.1 and two views that are "decomposed" from Q_2:

$$v_1(X_1, X_3) \text{:- } r(X_1, X_2), r(X_2, X_3).$$
$$v_2(X_1, X_3) \text{:- } r(X_3, X_4), r(X_4, X_5), r(X_5, X_1).$$

The following is an equivalent rewriting of Q_1 using the views:

$$Q_1() \text{:- } v_1(X_1, X_3), v_2(X_1, X_3), X_1 < X_3.$$

Notice the comparison $X_1 < X_3$ looks quite "different" from the comparison $X_1 < X_2$ in Q_1.

Example 2.4. This example shows that arithmetic comparisons could "export" nondistinguished variables. Consider the following query Q_1, and views v_1 and v_2:

$$Q_1(A) \text{:- } r(A), A \leqslant 4.$$
$$v_1(Y, Z) \text{:- } r(X), s(Y, Z), Y \leqslant X, X \leqslant Z.$$
$$v_2(Y, Z) \text{:- } r(X), s(Y, Z), Y \leqslant X, X < Z.$$

The following query P is a contained rewriting of the query Q_1 using v_1:

$$P(A) \text{:- } v_1(A, A), A \leqslant 4.$$

To see why, suppose we expand this query by replacing the view subgoal $v_1(A, A)$ by its definition. We get the expansion of P:

$$P^{\text{exp}}(A) \text{:- } r(X), s(A, A), A \leqslant X, X \leqslant A, A \leqslant 4.$$

The arithmetic comparisons imply $X = A$, and the expansion is thus contained in Q_1. Notice how the presence of the arithmetic comparisons helps in the existence of the rewriting. To see that, consider how the two views differ. Although v_1 and v_2 differ only in their second inequalities, v_2 cannot be used to answer Q_1. The reason is that the variable X of $r(X)$ in v_2 does not appear in the head, and it cannot be equated to another view variable appearing in the head using arithmetic comparisons. Therefore, the condition $A \leqslant 4$ in the query cannot be enforced on v_2. However, in v_1 the variable X of $r(X)$ was "exported" as distinguished with the help of the proper inequalities.

Example 2.5. This example shows the importance of the language of MCRs. For the following query and views, in the language of unions of CQACs, there is no MCR. We might need the power of Datalog to find a MCR:

$$\begin{aligned} &Q_2() \text{:- } e(X, Z), e(Z, Y), X > 6, Y < 8.\\ &v_1(X, Y) \text{:- } e(X, Z), e(Z, Y), Z > 6.\\ &v_2(X, Y) \text{:- } e(X, Z), e(Z, Y), Z < 8.\\ &v_3(X, Y) \text{:- } e(X, Z_1), e(Z_1, Z_2), e(Z_2, Z_3), e(Z_3, Y). \end{aligned}$$

We can show that for any positive integer $k > 0$, the following is a contained rewriting:

$$P_k \text{:- } v_1(X, Z_1), v_3(Z_1, Z_2), v_3(Z_2, Z_3), \ldots, v_3(Z_{k-1}, Z_k), v_2(Z_k, Y).$$

In fact, the following *recursive* Datalog program is a contained rewriting of the query:

$$\begin{aligned} &Q_2() \text{:- } v_1(X, W), T(W, Z), v_2(Z, Y).\\ &T(W, W) \text{:- } .\\ &T(W, Z) \text{:- } T(W, U), v_3(U, Z). \end{aligned}$$

This example shows that we may need a language more expressive than that of the query the views to have an MCR.

Several algorithms have been developed for answering queries using views, such as the bucket algorithm [28,21], the inverse-rule algorithm [32,18], and the algorithms in [6,30,31,2,27,1]. It has been shown that the problem of finding a rewriting of a query using views is $\mathcal{NP}$-complete, even if the query and the views are conjunctive [27] and the rewriting is expressed in the language of conjunctive queries.

Abiteboul and Duschka [1] use *certain answers* to denote those answers to the query that are contained in the answers of any database D over the database schema such that the following holds: the given view answers are among the output tuples when we apply the view definitions to this database D. Abiteboul and Duschka have also proven that, when both query and views are conjunctive, the maximal set of certain answers is obtained by maximally rewriting the query using the views (supposing an MCR exists) and then evaluating the rewriting using the views. Duschka [17] extends this result to the case where both the query and views are CQACs. In this paper, we focus on finding such rewritings. Note that the result in [1] is proven supposing a maximal rewriting exists. As we will see later, it is not easy to tell whether such a maximal rewriting exists, and moreover, it is hard to know how to find one.

3. Decidability results for the language of union of CQACs

In this section, we study the decidability of finding equivalent rewritings and MCRs for a query and views with respect to the language of union of CQACs.

3.1. Decidability result for equivalent rewritings

Theorem 3.1 (*CQAC equivalent rewriting*). *For a query and views that are CQACs, it is decidable whether there is an equivalent rewriting for the query using the views, in the language of rewritings that is conjunctive queries with comparisons. If such an equivalent rewriting exists, there is an algorithm to find it.*

Proof. The key idea is to compare a CQAC query Q with the expansion E of an equivalent rewriting P that is a single CQAC. Suppose Q is of size s. We consider all (at most $2^{O(s)}$) orderings of the variables and constants of Q that satisfy the arithmetic comparisons in Q. For each total ordering, there must be a containment mapping from E to Q that preserves order. Associate with each variable, V, of E a list of the $2^{O(s)}$ variables that are the images of V under each of these mappings. We define two variables of E as "equivalent" if their lists are the same. Since lists are of length at most $2^{O(s)}$ and each entry on the list has one of s values, there are at most $s^{2^{O(s)}}$ equivalence classes.

Design a new solution P' that equates all equivalent variables. P' is surely contained in P after expansion, since all we did was equate variables, thus restricting P and E. However, E', the expansion of P', has containment mappings to Q for all orderings, since all we did was equate variables that always went to the same variable of Q anyway. Thus Q is contained in P'. Since Q contains E, which contains E', it is also true that E' is contained in Q. Thus, P' is another equivalent rewriting of Q. Thus, there is a doubly exponential bound on the number of subgoals in P'. The conclusion is that we need to look only at some doubly exponentially sized solutions. □

This proof gives an exhaustive algorithm, and its search space is doubly exponential.

Theorem 3.2 (*union-of-CQAC equivalent rewriting*). *For a query and views that are CQACs, it is decidable whether there is an equivalent rewriting for the query using the views, where the rewriting is a finite union of conjunctive queries with comparisons. If such an equivalent rewriting exists, there is an algorithm to find it.*

Proof. We extend the proof of Theorem 3.1 to the case where an equivalent rewriting is a union of CQACs. Let P be a union of CQACs that is an equivalent rewriting of Q. We consider all orderings of the variables in Q that satisfy the arithmetic comparisons in Q. Now, however, for each ordering, there must be a containment mapping from the expansion of one of the CQACs of P to Q that preserves the order. Then, for each CQAC in P, we argue as in the proof of Theorem 3.1 to show that we need to look only at doubly exponentially sized solutions for each CQAC of P. Finally, there are only triply exponentially many combinations of CQACs of at most doubly exponentially size. We need to look at all of them. □

This proof gives an exhaustive algorithm, and its search space is triply exponential.

3.2. Decidability results for MCRs

Now we turn our attention to MCRs. We ask the following decidability question: for a given query and views in the language of conjunctive queries with comparisons, is there an MCR in the language of finite union of conjunctive queries with comparisons?

The proof in Theorem 3.1 is based on the fact that the query is contained in the rewriting's expansion. This fact puts a bound on the size of the rewriting, as the size of the query is given. In the case of MCRs, however, we cannot use this technique. In the presence of arithmetic comparisons, the containment test could use more than one containment mapping from the containing query to the contained one, unlike the case where pure conjunctive queries are involved. Therefore, potentially we might have to use an arbitrarily large number of mappings to test containment from the query to the expansion of the rewriting. Consequently, we might get arbitrarily long CQAC contained rewritings. In this section, we prove MCR, decidability for special cases by setting a bound on the size of a CQAC rewriting.

3.2.1. Views with no nondistinguished variables

We consider views that do not use nondistinguished variables in their definition, i.e., all variables used are also projected in the head.

Theorem 3.3 (*MCRs*). *Given a CQAC query and a set of CQAC views, where all view variables are distinguished. It is decidable whether there is an MCR of the query using the views w.r.t. the language of unions of CQACs*; *and there is an algorithm to find it.*

Proof. Let $Q = Q_0 + \beta_0$ be a CQAC, and V be a set of CQAC views. Suppose there is an MCR that is a union of CQACs using the views. Consider each CQAC P_j in the MCR, and P_j is a contained rewriting of Q.

The proof has two steps. In the first step, we replace each P_j by a set of rewritings whose union is equivalent to P_j, such that the arithmetic comparisons of each new rewriting define a *total* ordering on all its variables and constants. In the second step, we treat (after the modifications in the first step) the MCR as a union of CQACs, where the arithmetic comparisons in each CQAC define a total ordering. We consider each of these CQACs and show that its size is bounded. The second step is feasible because there are no nondistinguished variables in the view definitions, and the total ordering on the variables of a CQAC contained rewriting implies a total ordering on the variables of its expansion too.

First step: We replace P_j with a set $\{P_1^j \cdots P_{r_j}^j\}$ of contained rewritings whose union is equivalent to P_j as follows. For each ordering o_i of the variables and constants appearing in the views of P_j that satisfy its arithmetic comparisons, we construct a P_i^j that has the same ordinary subgoals as P_j and arithmetic comparisons that define the particular total ordering o_i on the variables and constants.

Second step: We consider a CQAC P of the MCR after step 1. Let $P = P_1 + \beta_1$, where P_1 uses P's ordinary subgoals and head, and β_1 is the arithmetic comparisons defining a total ordering of variables and constants appearing in P_1. Since all view variables are distinguished, we have $P^{\text{exp}} = P_1^{\text{exp}} + \beta_1$, and P^{exp} has exactly the same variables as P, hence, β_1 defines a total ordering on the variables and constants of P_1^{exp} too. For each P, we construct a new contained rewriting P' as follows. Since $P^{\text{exp}} \sqsubseteq Q$, by Lemma 2.1, there is a *single* containment mapping μ from Q to P^{exp}, such that $\beta_1 \Rightarrow \mu(\beta_0)$. The ordinary subgoals of P' are those views whose expansions contain subgoals in $\mu(Q_0)$. Its arithmetic comparisons are the projection of β_1 onto the variables in $\mu(Q_0)$. Notice that as all view variables are distinguished, there are no variables in $\mu(Q_0)$ that are not contained in P. We replace P by P'.

It remains to be proven that P' contains P and that P' is a contained rewriting of the query. P' contains P since P' has a subset of the subgoals of P. In addition, the containment mapping μ shows that the expansion of P' is contained in Q, since the expansion keeps the images of Q under μ. Moreover $\beta_1 \Rightarrow \beta_0$, and $\mu(AC(P'))$ is the projection of β_1 onto the variables in $\mu(Q_0)$. Since the query is safe, all variables in β_0 appear in Q_0. Thus, P' is a more containing contained rewriting of Q than P. Notice that the number of ordinary subgoals in P' is bounded by the number of ordinary subgoals in Q. Hence, there is a bound on the number of subgoals in P', and we need to look only at rewritings within this bound.

The number of view homomorphisms that we need to consider is exponential and the number of combinations of views that produce candidate rewritings is doubly exponential on the size of the input (the size of the input is equal to the size of the query and the size of the views). □

3.2.2. MCRs and AC-containment

Before we proceed with the next result, we discuss, in this subsection, the notion of two rewritings containing each other. We show that we need a subtler notion of containment between two rewritings in order to avoid arbitrarily long MCRs. Thus, we introduce here the notion of *AC-extension* of a rewriting and the notion of *AC-containment* between two rewritings, which leads to the notion of *AC-MCR*.

In the previous subsection, we were considering views with all variables distinguished, and we showed that for any contained rewriting there is a contained rewriting of bounded size which contains it. However, in general this is not the case as the following example shows.

Example 3.1. Consider the following query and views:

$$Q(A)\text{:- } r(A),\ A < 4.$$
$$v_1(Y, Z)\text{:- } r(X),\ s(Y, Z).$$
$$v_2(Y, Z)\text{:- } r(X),\ s(Y, Z),\ Y \leqslant X,\ X \leqslant Z.$$

We observe that the following is a rewriting:

$$P(Y_1)\text{:- } v_2(Y_1, Z_1),\ v_2(Y_2, Z_2),\ Z_1 \leqslant Y_2,\ Y_1 \geqslant Z_2,\ Y_1 < 4.$$

The expansion of P is

$$P(Y_1)\text{:-}\ r(X_1), s(Y_1, Z_1), Y_1 \leqslant X_1, X_1 \leqslant Z_1, r(X_2), s(Y_2, Z_2), Y_2 \leqslant X_2, X_2 \leqslant Z_2, Z_1 \leqslant Y_2, Y_1 \geqslant Z_2, Y_1 < 4.$$

We observe that in the expansion of P, all the variables in P will be equated because the two copies X_1 and X_2 of the nondistinguished variable in the view definition will be combined with the comparison subgoals in the rewriting and yield the equation. It is not hard to see that P is not contained in any rewriting that uses only one copy of the view although there is such a rewriting: $P'(X) : -v_2(X, X), X < 4$. However, rewriting P' cannot be obtained from P by standard tableau minimization, it does not suffice to remove subgoals, but we have also to add comparisons. For the same reason, for any positive integer k, the following is a rewriting:

$$P_k(Y_1)\text{:-}\ v_2(Y_1, Z_1), v_2(Y_2, Z_2), \ldots, v_2(Y_k, Z_k), Z_1 \leqslant Y_2, Z_2 \leqslant Y_3, \ldots, Z_{k-1} \leqslant Y_k, Z_k \leqslant Y_1, Y_1 < 4.$$

Moreover, there is no "shorter" rewriting that contains it.

In this example an MCR can be arbitrarily large. However, rewriting P_k is pathological in that, whenever there is a view instance on which the body of this rewriting is satisfied, all the variables in P_k are instantiated to the same constant and from this observation, it can be shown that a shorter rewriting can also serve to obtain the same answer to the query.

Thus, this example shows that a rewriting may have many semantically equivalent yet syntactically different variants, whose size is not a priori bounded. However, the "minimized" variants do have bounded size. The interesting part is that for the minimization, as opposed to known minimization techniques (e.g., tableau minimization), it does not suffice to simply remove subgoals, but one may have to also add comparisons. This is the reason AC-extensions are of interest.

Definition 3.1 (*AC-extension*). Let $\mathcal{V}$ be a set of views and P be a CQAC query using $\mathcal{V}$. The *AC-extension* of P is a query P' on $\mathcal{V}$ which is a copy of P with some additional arithmetic comparisons of the form $X\ \theta\ Y$ where X and Y are variables in P, and the expansion of P contains arithmetic comparison subgoals that imply $X\ \theta\ Y$.

Proposition 3.1. *Given a query Q, a view set $\mathcal{V}$, and a view instance I such that $I \subseteq \mathcal{V}(D)$ (for some D), let P be a rewriting and P' its AC-extension. Then P and P' produce the same set of answers on I.*

Proof. The one direction is easy because P contains P'. Let t be an answer to P. Then, the variable assignment that produced t in P can also serve as a variable assignment to produce t in P' because the additional comparison subgoals of P' are satisfied as a consequence of the fact that the constants in I satisfy the inequalities from the expansion of P (since $I \subseteq \mathcal{V}(D)$). Therefore, t is also an answer to P'. □

Definition 3.2 (*AC containment*). Let $\mathcal{V}$ be a set of views defined by CQACs and let P_1 and P_2 be two queries on $\mathcal{V}$. Let P_1' and P_2' be their AC-extensions. We say that P_1 *AC-contains* P_2 if P_1' contains P_2'.

In the example above, P_k is AC-contained in P, which is AC-contained in $P_0(A)$:- $v_2(A, A), A < 4$. Note that in order to decide AC-containment, we use the AC-extension of rewriting P that does not introduce any fresh variables; it only uses some additional comparisons among the variables already occurring in P. Hence it is not the same as containment as expansions. Therefore, it is applicable under the open-world assumption [1] because of Proposition 3.1.

Definition 3.3 (*AC-MCR*). Given a query Q and a view set $\mathcal{V}$, we call P an AC-MCR of Q w.r.t. $\mathcal{L}$ if (1) P is a contained rewriting (in $\mathcal{L}$) of Q, and (2) there is no contained rewriting P_1 (in $\mathcal{L}$) of Q such that P_1 properly AC-contains P.

Proposition 3.2. *Given a query Q and a view set $\mathcal{V}$ and a view instance I such that $I \subseteq \mathcal{V}(D)$. Let P be an AC-MCR and P^0 be an MCR over the language of union-CQAC (not necessarily finite). Then P and P^0 produce the same set of answers on I.*

Proof. The proof is a direct consequence of Proposition 3.1. □

3.2.3. Homomorphism property

The crux of the problem of rewriting conjunctive queries using views lies in ensuring that the expansion of the rewritten query is contained in the original query. Testing for containment of CQACs can be done more efficiently when the *homomorphism property* holds. Given a CQAC query Q, we denote by $core(Q)$ the ordinary (relational) subgoals of Q and by $AC(Q)$ the arithmetic comparison subgoals of Q.

Definition 3.4 (*homomorphism property*). Let $\mathcal{Q}_1$, $\mathcal{Q}_2$ be two classes of CQAC queries. We say that containment testing *on the pair* $(\mathcal{Q}_1, \mathcal{Q}_2)$ *has the homomorphism property* if for any pair of queries (Q_1, Q_2) with $Q_1 \in \mathcal{Q}_1$ and $Q_2 \in \mathcal{Q}_2$, the following holds: $Q_2 \sqsubseteq Q_1$ iff there is a homomorphism μ from $core(Q_1)$ to $core(Q_2)$ such that $AC(Q_2) \Rightarrow \mu(AC(Q_1))$.

In this case, we may apply the following containment test. The query q is contained in the query q' iff there is a mapping μ from the variables of q' to the variables of q such that (1) for the ordinary subgoals, μ is a containment mapping and (2) an arithmetic comparison subgoal $X\,\theta\,c$ maps to an arithmetic comparison subgoal $\mu(X)\,\theta\,c$. (For this test to hold, we assume that the ACs do not imply equalities and that the ACs of the contained query are complete, i.e., all the arithmetic comparisons that are implied by the ACs and use constants in the ACs of the containing query are computed. The latter is only a convenience, because, otherwise, we could say that each inequality of q' is mapped on an inequality which is implied by the ACs in q [5].)

Definition 3.5 (*homomorphism property for query rewriting*). Let $\mathcal{Q}_1$, $\mathcal{Q}_2$ be two classes of queries. We say that query rewriting problem *on the pair* $(\mathcal{Q}_1, \mathcal{Q}_2)$ *has the homomorphism property* if for any query $Q \in \mathcal{Q}_1$ and set of views $\mathcal{V} \in \mathcal{Q}_2$, the following holds: any rewriting (in the language of unions of CQACs) of Q using the views in $\mathcal{V}$ is such that its expansion can be tested for containment in the query by using a single containment mapping.

In cases where the homomorphism property holds, we have the following nondeterministically polynomial algorithm that checks if $Q_2 \sqsubseteq Q_1$. Guess a mapping μ from $core(Q_1)$ to $core(Q_2)$ and check whether μ is a containment mapping with respect to the AC subgoals too (i.e., an AC subgoal g maps on an AC subgoal g' so that $g' \Rightarrow g$ holds).

Klug [24] has shown that for the class of conjunctive queries with only open-LSI (open-RSI, respectively) comparisons, the homomorphism property holds. In [5] more cases are found where the homomorphism property holds. In [5] it is proven that in many natural cases of query and views where the query uses only LSI or only RSI comparisons the homomorphism property holds. The following theorem is an immediate consequence. It can be extended to capture a wider class of queries and views but if we do so, its statement will be somewhat cumbersome. [1]

Theorem 3.4. *In the following cases, the homomorphism property holds for the query rewriting problem*:

- *The query is an open-left-semi-interval* (*OLSI*) *conjunctive query* (*correspondingly open-right-semi-interval, i.e., ORSI*) *and the views are conjunctive queries with open arithmetic comparisons* (*CQOAC*).
- *The query is a closed-left-semi-interval* (*CLSI*) *conjunctive query* (*correspondingly closed-right-semi-interval, i.e., CRSI*) *and the views are CQAC*.

Now we present the third main result of this section in Theorem 3.5, which is an immediate consequence of the following proposition.

Proposition 3.3. *Let Q and V be a query and a set of views such that the homomorphism property holds for the query rewriting problem. Then for any contained rewriting P, there exists a contained rewriting P_1 which AC-contains P and the number of subgoals in P_1 is at most equal to the number of subgoals in the query.*

Proof. Consider the AC-extension P_e of P and its expansion $P_e^{\exp}$. Both the query and the expansion have been rewritten equivalently so that no equalities are implied by the ACs. Since the homomorphism property holds, there is a containment mapping μ that maps all subgoals (ordinary and comparison subgoals) of Q to subgoals in $P_e^{\exp}$. Now the key observation is that there is no pair of variables in P_e that are equated in $P_e^{\exp}$—the reason is that all ACs that

[1] Full details are given in [5].

would contribute to such an equation are already exported in P_e by definition. Thus, all variables that are targets of μ in $P_e^{\exp}$ appear in at most n subgoals in P_e (n is the number of subgoals in the query). Hence, we construct a rewriting P_1 by keeping those subgoals of P_e which contain target variables. It is easy to prove that P_1 is a contained rewriting and also contains P_e hence AC-contains P. □

Theorem 3.5 (*MCRs*). *Let Q and V be a query and a set of views such that the homomorphism property holds for the query rewriting problem. Then, there is an AC-MCR in the language of union of CQACs. Moreover, there is an algorithm to find it.*

In Section 5, we will provide an efficient algorithm to find an MCR in this case. Our algorithm extends the algorithm in [30,31] to capture comparisons in an efficient way.

4. Finding an MCR for queries using views without comparisons

In this section, we revisit the problem of finding an MCR for a query using views, where both the query and views do *not* have comparisons. We outline the MiniCon [31] and the shared-variable-bucket [30] algorithms to illustrate how they rewrite queries without arithmetic comparisons using views. Since these two algorithms are essentially similar, they are denoted "the MS algorithm" in the rest of this paper. Our algorithm extends the MS algorithm to handle arithmetic comparisons, and the proof of the correctness of our algorithm is an extension of the correctness proof of the MS algorithm. Thus, we give a complete description of the MS algorithm together with the proof for completeness and soundness. Then, in Section 5, based on the description of the MS algorithm, we first point out the complications introduced by the presence of arithmetic comparisons. We then present our algorithm and prove its completeness and soundness. Most of the techniques developed in these two sections is used to prove completeness and soundness.

4.1. Mappings and the most containing rewriting

4.1.1. Motivating example

Our setting consists of a conjunctive query and a set of conjunctive views. We name the subgoals in the query and the view definitions by unique names. If there is a subgoal $X = Y$, equating variables X and Y, then we replace variable Y by X and delete the equation from the subgoals. A rewriting might have multiple occurrences of the same view. Although we retain the same view subgoal name for different occurrences of a view, we may use a new set of variable names, reflecting the fact that in the expansion of a rewriting we use fresh variables for each occurrence of a view.

Example 4.1. Consider three relations: relation `car(make, dealer)` stores information about car makes and dealers who sell them. Relation `loc(dealer, city)` stores information about dealers and their located cities. Relation `part(store, make, city)` has information about a store, the car makes whose parts are sold by the store, and the store's located city. A user submits the following query:

$$Q : q_1(S, C) \text{:- } car(M, anderson), loc(anderson, C), part(S, M, C)$$

which asks for cities and stores that sell parts for car-makes sold in the `anderson` branch in this city.

Assume that we have the following views on the base relations, and we need to consider two occurrences of view V_1. (For each occurrence of a view in a rewriting, the MS algorithm chooses a copy of the view. Here, for the sake of an example, we show arbitrarily two copies of V_1.)

$$V_1 \text{:- } v_1(M_1, D_1, C_1) \text{:- } car(M_1, D_1), loc(D_1, C_1).$$
$$V_1' \text{:- } v_1(M_1', D_1', C_1') \text{:- } car(M_1', D_1'), loc(D_1', C_1').$$
$$V_2 \text{:- } v_2(S_2, M_2, C_2) \text{:- } part(S_2, M_2, C_2).$$
$$V_3 \text{:- } v_3(M_3, D_3, C_3) \text{:- } car(M_3, D_3), loc(D_3, C_3).$$

We name the three subgoals of the query by g_1, g_2, and g_3, respectively. We name the first subgoal of view v_1 by g_{11} and the second subgoal g_{12} and, in general, the jth subgoal of view v_i by g_{ij}.

Let P be a contained rewriting of Q using the views. Then, there is a containment mapping from Q to the expansion P^{exp} of P, which proves that P^{exp} is contained in Q. This containment mapping can be viewed as a *subgoal mapping* from subgoals of Q to subgoals of the views that P is using, together with an *argument mapping* among the variables and constants used in the arguments of those subgoals. (The MS algorithm first considers subgoal mappings, and then argument mappings, and finally checks whether the mappings can be turned to containment mapping.) Now consider the rewritings:

$$P_1\text{:- } q_1(S, C)\text{:- } v_1(M, \mathit{anderson}, C), v_2(S, M, C).$$
$$P_2\text{:- } q_1(S, C)\text{:- } v_1(M, \mathit{anderson}, C_1), v_1(M_1', \mathit{anderson}, C), v_2(S, M, C).$$

For rewriting P_1, the containment mapping from the query to the expansion of the rewriting can be viewed as (a) the subgoal mapping: g_1 to g_{11}, g_2 to g_{12}, and g_3 to g_{21}; (b) the argument mapping: M to M_1, *anderson* to D_1, C to C_1, S to S_2, M to M_2, and C to C_2. For rewriting P_2, the subgoal mapping is the same. However (since we use two occurrences of view v_1), the argument mapping is: M to M_1, *anderson* to D_1, *anderson* to D_1', C to C_1', S to S_2, M to M_2, and C to C_2. For rewriting P_2, we say subgoal g_1 is *covered* by g_{11}, g_2 is covered by g_{12}, and g_3 is covered by g_{21}.

4.1.2. Mappings and contained rewritings

Based on this intuition, we define three kinds of mappings for a query and a set of views.

A *subgoal mapping* is a mapping from the query subgoals to view subgoals of a view such that the predicate names match. A subgoal mapping is *total* if it maps all query subgoals.

A subgoal mapping induces an *associated argument mapping* that maps each query variable/constant to a variable/constant in the body of the view definition, such that for each query subgoal g that is mapped to a view subgoal, their variables and constants are also mapped argument-wise. (For each query subgoal, we use a fresh copy of a view.) Notice that an argument mapping is not restricted to map a query variable/constant to a single view variable/constant (as in a containment mapping), since it may map a query variable/constant to several view variables/constants.

Given an argument mapping, we associate with it several *containment mappings*. An *associated containment mapping* is a mapping from query variables/constants to view variables/constants defined by a partition $\mathcal{P}$ on the set of the view variables/constants into equivalence classes, in such a way that: (1) Each query variable/constant is mapped to elements of a single equivalence class. (2) The following three conditions hold: (a) each equivalence class with more than one element is populated by either (identical) constants or/and distinguished variables; (b) an equivalence class that is the image of a constant has only distinguished variables (even if it contains only one element) and possibly the same constant; (c) Distinguished variables map to distinguished variables. (3) All variables/constants of a query subgoal are mapped to the variables/constants of a single copy of a view. By extension, we define an associated containment mapping of a subgoal mapping.

Given a total subgoal mapping and one of its associated containment mappings $\mathcal{M}$ (if there exists any), we define the following query over view subgoals. The defined query is the one that uses the view copies that are involved in the associated containment mapping. Distinguished view variables are equated according to the partition that defines the associated containment mapping. We call this query the *associated view query* or *associated query rewriting* of the containment mapping $\mathcal{M}$.

Proposition 4.1. *Given a total subgoal mapping and an associated containment mapping $\mathcal{M}$ of it, the associated view query of $\mathcal{M}$ is a contained rewriting.*

Proof. It is easy to prove that the associated containment mapping is a containment mapping from the query to the expansion of the rewriting. □

Thus, we can refer to this contained rewriting as the *associated contained rewriting* of $\mathcal{M}$. Moreover, considering a total subgoal mapping and all its associated containment mappings, we refer to all associated contained rewritings as the *associated contained rewritings* of the subgoal mapping.

Now we show that each contained rewriting is produced as an associated rewriting of a subgoal mapping.

Proposition 4.2. *Given a contained rewriting P, there is a subgoal mapping and an associated containment mapping such that P is the associated rewriting of this containment mapping.*

Proof. Take the expansion of P and the containment mapping from the query to the expansion that proves that P is a contained rewriting. This containment mapping induces a subgoal mapping and an associated containment mapping. □

Example 4.2. In our running example, let us consider rewriting P_2 and the subgoal mapping that produces it (as in Proposition 4.2). Taking the argument mapping of this subgoal mapping, we also consider the associated containment mappings. First, we observe that there is more than one containment mapping associated with this argument mapping. In fact, one of those associated containment mappings is a containment mapping associated with P_2 and another with P_1. The following two partitions are associated containment mappings:

- Partition $\mathcal{M}_1$ has three equivalence classes: $\{D_1, D_1'\}$, $\{M_1, M_2, M_1'\}$, and $\{C_1, C_2, C_1'\}$.
- Partition $\mathcal{M}_2$ has five equivalence classes: $\{D_1, D_1'\}$, $\{M_1, M_2\}$, $\{C_1', C_2\}$, $\{M_1'\}$, and $\{C_1\}$.

In the first mapping, the two occurrences of view v_1 are identical. Hence we delete one occurrence and get rewriting P_1. The second mapping $\mathcal{M}_2$ constructs rewriting P_2. Observe that P_1 is contained in P_2 as queries. Thus, $\mathcal{M}_1$ is contained in $\mathcal{M}_2$.

So far we have settled that in order to find all rewritings, it suffices to consider all total subgoal mappings, and for each subgoal mapping, find all its associated rewritings. Now we prove that, when we want to construct a MCR, for each subgoal mapping, we only need to construct one associated rewriting. The reason is that all other associated rewritings of this subgoal mapping are contained in this one. We shall call this rewriting the most relaxed (or the most containing) rewriting of this subgoal mapping.

4.1.3. The most containing (relaxed) rewriting

Given a specific argument mapping, we say that a containment mapping $\mathcal{M}_1$ *contains* a containment mapping $\mathcal{M}_2$ if the partition that defines $\mathcal{M}_1$ "contains" the partition that defines $\mathcal{M}_2$, i.e., any equivalence class of the second is the union of some equivalence classes of the first (also known as the one partition being a *finer* partition of the other).

Proposition 4.3. *Consider a total subgoal mapping and two associated containment mappings $\mathcal{M}_1$ and $\mathcal{M}_2$. Then $\mathcal{M}_1$ contains $\mathcal{M}_2$ iff the associated contained rewriting of $\mathcal{M}_1$ contains the associated contained rewriting of $\mathcal{M}_2$.*

Lemma 4.1. *Let M be a subgoal mapping and let $\mathcal{R}$ be all the associated containment mappings. Then all containment mappings in $\mathcal{R}$ form a semi-lattice with respect to partition containment.*

Proof. We need to prove that for any pair of P_1 and P_2 in $\mathcal{R}$, there exists a containment mapping P in $\mathcal{R}$ such that (a) P contains both P_1 and P_2; and (b) P is contained in any associated mapping in $\mathcal{R}$ that contains both P_1 and P_2. The associated containment mapping P is defined by the intersection partition of the partitions that define P_1 and P_2. The intersection partition is defined by taking as equivalence classes all pairwise intersections of an equivalence class in P_1 with an equivalence class in P_2.

First, we prove that P is an associated containment mapping in $\mathcal{R}$. We prove that each query variable has its images in a single equivalence class. Suppose that query variable X is mapped to variables in two distinct equivalence classes of P. Then, X is either mapped to two distinct equivalence classes in P_1, or mapped to two distinct equivalence classes in P_2. This result contradicts the fact that X maps to a single equivalence class in P_1 (P_2, respectively). To prove (a): The containment mapping from P to P_1 is defined by mapping all variables in an equivalence class of P to the equivalence class of P_1 they were constructed from. To prove (b): Let P' be the associated containment mapping that contains both P_1 and P_2. Hence, each equivalence class in P' is contained in an equivalence class C_1 of P_1 and in an equivalence class C_2 of P_2. Therefore, it is also contained in the intersection of C_1 and C_2 which is an equivalence class of P. □

Lemma 4.2. *Let M be a subgoal mapping and let $\mathcal{P}$ be all the associated contained rewritings. Then all rewritings in $\mathcal{P}$ form a semi-lattice with respect to query containment.*

Proof. The proof is a consequence of Lemma 4.1. □

Input: a subgoal mapping
Output: the most relaxed associated containment mapping
Method:
For each query variable/constant X do:
Form a class that contains all view variables/constants that are images of X;
While classes are not disjoint do:
Merge classes that share an element;
return(classes) or fail if there is a class containing two distinct constants;

Fig. 2. Procedure: FindMostRelaxedMapping.

Corollay 4.1. *Given a total subgoal mapping, there exists an associated rewriting that contains all associated rewritings of this subgoal mapping. We call the rewriting the* most relaxed rewriting, *and the corresponding containment mapping* most relaxed containment mapping.

For a given subgoal mapping, the above results show a semi-lattice structure of the containment relationship among associated containment mappings and a semi-lattice structure of the containment relationship among associated rewritings. For each subgoal mapping, it suffices to consider the most relaxed rewriting, since the rest are contained in it. In conclusion, we have proven so far that an algorithm that considers all subgoal mappings and for each subgoal mapping computes the most relaxed rewriting (if there exists one) is complete.

4.2. The MS algorithm

Now we formally present the MS algorithm and prove its correctness. So far we have shown that, to save on the number of rewritings in the MCR, for each subgoal mapping, we only need to consider the most relaxed rewriting, since all other associated rewritings are contained in it. The algorithm also prunes subgoal mappings that do not have any associated containment mapping early in the algorithm. We do the pruning by constructing subgoal mappings in a systematic fashion, then trying to construct associated containment mappings for subgoal mappings that are not necessarily total, and discard this branch if we fail. Based on condition (3) in the definition of containment mappings, the following is an easy but very useful observation towards formalizing this pruning.

Lemma 4.3. *A total subgoal mapping has at least one associated containment mapping only if it can be decomposed into* partial subgoal mappings, *each of which uses only one view copy and has the properties*: (*a*) *it has an associated containment mapping*; *and* (*b*) *if a query variable X is mapped to a nondistinguished view variable*, *then all query subgoals that contain X belong in this partial mapping*.

In this lemma, property (b) is called the *shared-variable property*. A partial subgoal mapping is called an MCD if it is minimal, i.e., it cannot be decomposed into other nontrivial partial mappings. (MCD stands for MiniCon description and is introduced in [31]). The decomposition property established in Lemma 4.3 is called the *local property*. Now the algorithm finds MCDs and combines them. Notice that in the MS algorithm (formally described next), we should still check in the end whether an associated containment mapping exists.

A *partial MCD with shared variables* is a subgoal mapping on a single view copy where the following is true:

1. There is a query subgoal in this partial MCD that contains a variable X mapped to a nondistinguished view variable; and
2. X also occurs in query subgoals that do not belong to this partial MCD—X is referred to as the *shared variable*.

We call a subgoal mapping *legal* if it has an associated containment mapping. A *legal* MCD is defined by a legal subgoal mapping. Before we describe the two parts of our algorithm, namely the two procedures "GenMCD" and "CombineMCD," we describe a procedure that is called in both to find legal MCDs and to find the most relaxed mapping when a subgoal mapping is given. This is the procedure "FindMostRelaxedMapping" (Fig. 2).

```
Input: a query Q, a set V of views;
Output: a set of MCDs;
Method:
for each query subgoal g in Q
    for each view V in V
        for each subgoal g' in V {
            find containment mapping μ from g to g', if there exists one;
            if μ exists, construct new partial MCD: (G = {g}, μ)
        }
while (there are partial MCDs with shared variables) {
    for each partial MCD (G, μ) {
        choose a shared variable X in G;
        choose a query subgoal g not in G and that contains X;
        for each view subgoal g' that has an argument mapping μ' from g {
            extend μ with μ';
            replace the current partial MCD with new partial MCD: (G' = G ∪ {g}, μ' = ext(μ));
        }
    }
}

return (all legal MCDs);
```

Fig. 3. Procedure: GenMCD.

```
Input: a set of MCDs
Output: a set of all most relaxed rewritings
Method:
    For each combination of MCDs that covers all query subgoals without overlapping {
        Check whether there exists an associated containment mapping;
        If it does {
            Find the most relaxed associated containment mapping;
            Create the most relaxed rewriting;
        }
    }
    return (these rewritings);
```

Fig. 4. Procedure: CombineMCD.

Proposition 4.4. *The above procedure produces the most relaxed associated containment mapping.*

Proof. In each step, a class is a subset of an equivalence class in any partition, such that an initial class (obtained by the argument mapping) is contained in an equivalence class. Since we stop merging classes as soon as we reach a phase where classes are disjoint, which means that we reach a partition, this is the finer partition. □

Let G_Q be the set of all query subgoals. The first step of the algorithm constructs MCDs, as shown in the procedure "GenMCD" in Fig. 3.

The second step of the algorithm combines MCDs to generate rewritings, as shown by the procedure "CombineMCD" in Fig. 4. We say that a set of MCDs $(G_1, \mu_1), \ldots, (G_m, \mu_m)$ *covers all query subgoals without overlapping* if the following conditions hold: (i) the pairwise intersection of the query subgoals set is the empty set, i.e., $G_i \cap G_j = \emptyset$ for all $i \neq j$; and (ii) the union of all query subgoals sets is equal to the set of all query subgoals,i.e., $G_1 \cup \ldots \cup G_m = G_Q$.

In the procedure, the reason we only consider nonoverlapping subgoal mappings will be clear in the soundness proof. The following theorem proves that the MS algorithm is sound and complete.

Theorem 4.1. *Given a query and views that are conjunctive queries, the MS algorithm finds an MCR in the language of union of conjunctive queries.*

Proof. *Completeness*: A straightforward consequence of Corollary 4.1 and Proposition 4.4.

Soundness: There exists a mapping from the query to the expansion of the rewriting. This is the union of all mappings associated with MCDs that were covered by views in the rewriting. It remains to prove that the union maps a query variable/constant to a single variable/constant in the expansion. Let a query variable X be mapped on a view variable Y in $MCD1$ and on a view variable Z in $MCD2$. If both Y and Z are distinguished view variables, then we can equate them. If one of those is nondistinguished (say Y), then all query subgoals containing X are in $MCD1$. As there is no overlapping, no query subgoal containing X is in $MCD2$ and as we take the most relaxed variable mapping for each MCD, X has no image under $MCD2$. This is a contradiction. Similarly, for a constant C in the query, by construction of the MCDs, constant C maps to either a distinguished variable Z or the same constant C. The distinguished variable Z is then replaced by the constant C in the rewriting. If two constants C and C' map to the same distinguished variable Z, then the algorithm rejects the mapping in the last step. □

5. Finding an MCR for queries using views with comparisons

In this section, we present an algorithm for finding an MCR for a query using views, where both the query and the views are CQACs. We assume the existence of the homomorphism property between the query and the expansion of each MCR. The following is a direct consequence of the results in [5] and the discussion in Section 3.2.3. The algorithm is applicable to the following cases:

- The query is OLSI conjunctive queries (correspondingly ORSI) and the views are CQOAC.
- The query is a CLSI conjunctive query. The views are CQAC.

As in the case without comparisons, our algorithm can be thought of as having two parts. The first part constructs buckets, and finds partial mappings from the query subgoals to the view subgoals. The second part combines these mappings to construct an MCR.

For the rest of this section, whenever we refer to contained rewritings, we mean the AC-extensions of contained rewritings, unless otherwise mentioned. The first subsection presents the new ideas that need to be introduced in the algorithm of the previous section in order for the algorithm to capture comparison subgoals as well. The second subsection contains the algorithm and the proof of correctness.

5.1. Exportable nondistinguished view variables

In this subsection, we develop our tools and show informally with examples why these technical notions are needed in our algorithm. The algorithm we develop in this section is an extension of the algorithm in the previous section and it has the same structure. So, in this subsection, while informally explaining the usability of the new notions, we refer to concepts we defined in Section 4. However, we will formally define again (when necessary) those concepts in Section 5.2, where we formally describe the algorithm.

Let us revisit Example 3.1, which shows that a nondistinguished view variable can be exported due to the comparison predicates in the views.

Example 5.1. Consider the following query and views:

$$Q(A)\text{:- } r(A),\ A < 4.$$
$$v_1(Y, Z)\text{:- } r(X),\ s(Y, Z).$$
$$v_2(Y, Z)\text{:- } r(X),\ s(Y, Z),\ Y \leqslant X,\ X \leqslant Z.$$

While trying to use v_1 to answer query subgoal $r(A)$, we have a partial mapping $A \to X$. However, variable A appears in $A < 4$, but X is a nondistinguished view variable. Since v_1 does not export variable X, we cannot put a restriction $X < 4$ on X in a rewriting that uses v_1 to cover $r(A)$. Thus, this partial mapping will be rejected in step 1 of the algorithm.

Even though v_2 has the same ordinary subgoals as v_1, we cannot reject the mapping from $r(A)$ to $r(X)$ in v_2. The reason is that we can *export* variable X due to its comparison predicates. In particular, the following is a contained

rewriting of the query using v_2:

$$Q(A)\text{:- } v_2(A, A), A < 4.$$

In this contained rewriting, we equate v_2's head variables Y and Z, and its comparison predicates become $A \leqslant X$ and $X \leqslant A$, implying that $A = X$. Then variable X becomes exported, and we can add $A < 4$ to the rewriting.

Another slightly different aspect of the same observation can be shown in the case of the following query

$$Q'\text{:- } r(A), A < 4.$$

Then we have the following rewriting:

$$Q'\text{:- } v_2(Y, Z), Z < 4.$$

The constraint "< 4" is imposed on the argument of r indirectly because it is implied (in the expansion of the rewriting) by the two inequalities $Z < 4$ (in the rewriting) and $X \leqslant Z$ (in the definition of the view).

Definition 5.1 (*exportable view variables*). A nondistinguished variable X in a view v is *exportable* if there are two distinguished view variables Y and Z, such that the equation $Y = Z$ together with the comparisons of the view imply that $X = Y = Z$. In this case, we say that variable X can be exported.

5.1.1. Conditions for exporting variables

To find exportable nondistinguished variables in a view v, we use the comparison predicates in v to construct its *inequality graph* [24], denoted $G(v)$. That is, for each comparison predicate $A\ \theta\ B$, where θ is $<$ or $\leqslant$, we introduce two nodes labeled A and B, and an edge labeled θ from A to B. Clearly if there is a path between two nodes A and C, we have $A < C$. If there is no $<$-labeled edge on any path between A and C, then $A \leqslant C$.

Definition 5.2 (*leq-set*). Given a nondistinguished variable X in a view v, the less-than-or-equal-to set (*leq-set*) of X, denoted $S_{\leqslant}(v, X)$, includes all distinguished variables Y of v that satisfy the following conditions: there exists a path from Y to X in the inequality graph $G(v)$, and all edges on all paths from Y to X are labeled $\leqslant$. In addition, in all paths from Y to X, there is no other distinguished variable except Y.

Correspondingly, we define the greater-than-or-equal-to set (*geq-set*) of a variable Y, denoted $S_{\geqslant}(v, Y)$. We want to know which view variables are exportable. For instance, in Example 3.1, $S_{\leqslant}(v_1, X) = \{\}$, $S_{\geqslant}(v_1, X) = \{\}$, $S_{\leqslant}(v_2, X) = \{Y\}$, and $S_{\geqslant}(v_2, X) = \{Z\}$.

Lemma 5.1. *A nondistinguished variable X in view v is exportable iff both $S_{\leqslant}(v, X)$ and $S_{\geqslant}(v, X)$ are nonempty.*

Proof. If the sets are nonempty, choose one element from each and equate them to obtain a head homomorphism h. X is exportable using h. If the variable is exportable, by definition, there are variables in the $S_{\leqslant}$ and the $S_{\geqslant}$. Thus, they are nonempty. □

To export a nondistinguished variable X in a view v, we can equate any pair of variables (Y_1, Y_2), where $Y_1 \in S_{\leqslant}(v, X)$ and $Y_2 \in S_{\geqslant}(v, X)$. X becomes exported since it is equal to Y_1 and Y_2, as are all variables in the path from Y_1 to Y_2.

Example 3.1 shows that comparison predicates make it possible to equate even nondistinguished variables. While constructing a partial mapping from a query subgoal g to a subgoal in view v, a query variable A might be mapped to two different view variables X_1 and X_2. These variables still could be equated, as illustrated by the following example.

Example 5.2. Consider the following query and views

$$Q(A)\text{:- } (r(A, A).$$
$$v(X_1, X_2, X_3, X_6, X_7, X_8)\text{:- } r(X_4, X_5), s(X_1, X_2, X_3, X_6, X_7, X_8), X_3 \leqslant X_5, X_5 \leqslant X_7, X_1 \leqslant X_4, X_8 \leqslant X_2,$$
$$X_2 \leqslant X_4, X_4 \leqslant X_6.$$

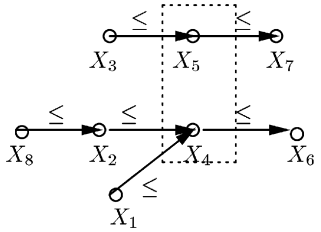

Fig. 5. The graph $G(v)$ in Example 5.2.

Fig . 5 shows the graph $G(v)$. In order to construct a mapping from query subgoal $r(A, A)$ to view subgoal $r(X_4, X_5)$, we need to equate X_4 and X_5, since both are the images of A. That is, we need $X_4 \leqslant X_5$ and $X_5 \leqslant X_4$. For the former, it can be satisfied if there is a path from X_4 to X_5 in graph $G(v)$. If such a path does not exist, we can have this inequality by equating a variable in $S_{\geqslant}(v, X_4)$ with a variable in $S_{\leqslant}(v, X_5)$. A similar argument holds for $X_5 \leqslant X_4$. Since neither inequality exists in the graph, we need to satisfy them by equating distinguished variables.

Clearly we have $S_{\leqslant}(v, X_5) = \{X_3\}$, $S_{\geqslant}(v, X_5) = \{X_7\}$, $S_{\leqslant}(v, X_4) = \{X_1, X_2\}$, and $S_{\geqslant}(v, X_4) = \{X_6\}$. Note that X_8 is *not* in $S_{\leqslant}(v, X_4)$, because X_2 is "closer" to X_4 in the path from X_8 to X_4. The following are two most relaxing ways to equate variables to imply $X_4 = X_5$: (1) $X_6 = X_3, X_1 = X_7$, and (2) $X_6 = X_3, X_2 = X_7$. They are most relaxing in the sense that any other way to equate variables to imply $X_4 = X_5$ either includes the comparisons in (1) or it includes the comparisons in (2).

In our algorithm, we construct a set P of pairs of view variables that should be equated, so as to construct a valid partial mapping. Note that we have to consider only *valid* equating of variables (similar to head homomorphisms in [31]). Namely, while equating variables to generate head homomorphisms for a view, some head homomorphisms make the comparison predicates in the view not satisfiable, and the view should be removed from the buckets. For instance, consider the following query and view:

$$Q(X, Y) \text{:- } p(X, Y), X < 3, Y > 5.$$
$$v(A) \text{:- } p(A, A).$$

We construct a mapping μ to map both X and Y to A. However, μ will map the query comparison predicates to "$A < 3$ and $A > 5$," which is not satisfiable. Thus, we cannot use this view to cover the query subgoal.

5.1.2. Dual roles of exportable nondistinguished variables

When a nondistinguished query variable maps to an exportable nondistinguished variable, we have two choices. Either we can export the nondistinguished variable and then treat it as a distinguished variable, or we can treat it as a nondistinguished variable and map to it. The following example illustrates the dual roles exportable nondistinguished variables can play.

Example 5.3. Consider the following query and views:

$$Q \text{:- } p(A), r(A).$$
$$v_1(X) \text{:- } r(X).$$
$$v_2(X, Z) \text{:- } p(X), r(Y), s(Y, Z), X \leqslant Y, Y \leqslant Z.$$

To cover the query subgoal $p(A)$, we need to use the view v_2. Since A maps to the nondistinguished Y, we can export Y first and then create a multi-subgoal bucket corresponding to the subgoals that share A, namely, $p(A)$ and $r(A)$. The view v_2 covers both subgoals and thus, we have the contained rewriting

$$R1 \text{:- } Q \text{:- } v_2(A, A).$$

Alternatively, we can use v_2 to cover p and v_1 to cover r and thus have the rewriting

$$R2 \text{:- } Q \text{:- } v_1(X), v_2(X, Z).$$

Observe that in rewriting $R1$ we have exported variable Y, whereas in rewriting $R2$ we did not need to export variable Y. Moreover, these two rewritings do not contain each other. Thus, although variable Y *can* be exported, if we restrict ourselves to obtaining only those rewritings in which Y is used as an exported variable, we miss some rewritings. The missed rewritings are not contained in any other rewritings that use Y as an exported variable. Therefore, variables (like Y) that can be exported must be used in our algorithm in both their roles, as variables that are exported and as variables that are treated as regular nondistinguished variables.

5.1.3. Satisfying comparisons in the rewriting

In the second step of our algorithm, we consider combinations of views from the buckets to answer all query subgoals. Each combination represents a candidate rewriting, and we add comparison predicates to satisfy the comparison predicates in the query. Consider a query arithmetic comparison "$X\ \theta\ c$," where X is mapped to a view variable Y in a partial mapping, and θ is $<$ or $\leqslant$. The expansion of a rewriting must imply the image of this restriction, i.e., $Y\ \theta\ c$. If Y is distinguished, we can just add "$Y\ \theta\ c$" to the rewriting. If Y is nondistinguished, we cannot add any arithmetic comparison using Y, since Y does not appear in the rewriting at all. However, there are two ways to satisfy this restriction even in the case that Y is nondistinguished.

Case I: The arithmetic comparisons of the view v imply "$Y\,\theta\,c$" by themselves.

Case II: There is a path in $G(v)$ from Y to a distinguished variable Z, so we can just add an arithmetic comparison "$Z < c$" or "$Z \leqslant c$" as appropriate to the rewriting to satisfy "$Y\,\theta\,c$."

For example, consider the following query and views:

$$
\begin{aligned}
&Q(A) \text{ :- } p(A), A < 3.\\
&v_1(X_1) \text{ :- } p(X_1), X_1 < 3.\\
&v_2(X_2, X_3) \text{ :- } p(X_1), r(X_2, X_3), X_2 \leqslant X_1, X_1 \leqslant X_3.\\
&v_3(X_2, X_3) \text{ :- } p(X_1), r(X_2, X_3, X_4), X_2 \leqslant X_1, X_3 \leqslant X_1, X_1 \leqslant X_4.
\end{aligned}
$$

While mapping the query subgoal $p(A)$ to the view subgoal $p(X_1)$ in view v_1, we have a partial mapping μ that maps variable A to X_1. For a rewriting of the query $Q(A)$ that uses this view, its expansion should entail $\mu(A < 3)$, i.e., $X_1 < 3$. The comparison predicate in v_1 belongs to case I, since its comparison predicate $X_1 < 3$ can satisfy this inequality. The comparison predicates in v_2 belong to case II. In particular, since v_2 has a comparison predicate $X_1 \leqslant X_3$, and X_3 is distinguished, thus we can add $X_3 < 3$ to satisfy the inequality $X_1 < 3$. The comparison predicates in v_3 do not belong to either case, thus v_3 cannot be used to cover the query subgoal.

5.2. Extending the MS algorithm to CQACs

Now we present formally our algorithm for generating MCRs for a query using views. Without loss of generality, we assume that the comparisons in the query and the views do not imply equalities.

5.2.1. Mappings and the most containing rewritings

First let us repeat the following definition. A *distinguishable or exportable variable* is a variable X such that there are two view variables X_1 and X_2 with a $\leqslant$-path from X_1 to X to X_2. We call X_1 and X_2 *anchors*. Later on, in describing the algorithm we will distinguish between distinguishable and exported variables, in that by "distinguishable" we will mean that are potentially able to be treated as distinguished, whereas by "exported" we will mean that we actually treat them as distinguished and add the necessary equalities to export them. A *semi-distinguishable variable* is a variable such that there is a θ-path from the variable to a distinguished variable. The latter variable is called the *anchor*. We say then that the variable has an anchor.

We will use the notions defined in Section 4.1.2 with a few changes. We will retain the first item of that definition that defines a subgoal mapping, and the second item that defines an argument mapping. However, we change the definition of an associated containment mapping slightly. In the definition that follows we "almost" repeat the third item in the definition of Section 4.1.2 with a few changes that are marked in emphasized font.

Definition 5.3 (*mappings*). Assume we are given a query and a set of views. We denote the conjunction of the ACs in the query by β_1. Given an argument mapping, we associate with it several AC-containment mappings. An *associated AC-containment mapping* is defined by a partition $\mathcal{P}$ on the set of the view variables/constants into equivalence classes

together with a set S_{AC} of inequalities on the view variables, in such a way that each query variable/constant is mapped to a single equivalence class, and the following three conditions hold:

(a) Each equivalence class with more than one element is populated by either (identical) constants or/and distinguished variables *or/and distinguishable variables*.
(b) An equivalence class that is the image of a constant has only distinguished *or distinguishable variables* (even if it contains only one element).
(c) Distinguished variables map to distinguished *or distinguishable variables*.
(d) *If a query variable X in β_1 (hence there is a comparison $X\,\theta\,c$) maps on an equivalence class, then this class contains distinguished or distinguishable or semi-distinguishable view variables and $Y\,\theta\,c$ is added to S_{AC}, where Y is the variable representing the equivalence class in the first two cases, and is the anchor of the class variable in the last case.*

By extension, we define an associated containment mapping of a subgoal mapping.

As in Section 4.1, we define the associated rewriting of an associated AC-containment mapping and we get the following two propositions that are the same as Propositions 4.1 and 4.2 (only with a slightly different proof).

Proposition 5.1. *Given a total subgoal mapping and an associated AC-containment mapping $\mathcal{M}$ of it, the associated view query of $\mathcal{M}$ is a contained rewriting.*

Proposition 5.2. *Given a contained rewriting P, there is a subgoal mapping and an associated AC-containment mapping such that P is the associated rewriting of this AC-containment mapping.*

Proof. The proof is along the same lines as Proposition 4.2. □

Thus, the above propositions have settled that a total AC-containment mapping produces a rewriting and vice versa.

5.2.2. The most containing rewritings

Now, we will discuss how to construct the most containing rewritings. Given a subgoal mapping, we define containment among its associated AC-containment mappings as in Section 4.1 only extending it to include that they use the same comparisons. Thus, we have again the following proposition.

Proposition 5.3. *Consider a total subgoal mapping and two AC-associated containment mappings $\mathcal{M}_1$ and $\mathcal{M}_2$. Then $\mathcal{M}_1$ contains $\mathcal{M}_2$ iff the associated contained rewriting of $\mathcal{M}_1$ contains the associated contained rewriting of $\mathcal{M}_2$.*

We are given an associated AC-containment mapping and the inequality graph. As we mentioned, the partition into equivalence classes has implications for some nondistinguished view variables due to the existence of the arithmetic comparison predicates.

An AC-containment mapping partition is maximal if there is no other AC-containment mapping partition that contains it. In the non-AC case, we proved that there is only one maximal containment mapping partition. Now we may have several.

In the case without comparisons, when we were to define a containment mapping, we were defining equivalence classes explicitly. Now, besides defining them explicitly, there is an implicit way that puts variables into classes. Whenever two variables belong to the same class and there is a third variable that is connected by comparisons to both, then these comparisons together with the equation of the two variables (implied by the fact that they belong to the same equivalence class) may imply that the third variable is also equal, and hence should be put in the same class. Note that this is a consequence of the fact that we understand an equivalence class, in this setting, as a set of variables that are equated. For example, suppose that variables X and Y are in the same class and there are two comparisons: $X \leqslant Z$ and $Y \geqslant Z$. Since the fact that X and Y are in the same equivalence class implies that $X = Y$, this equation together with the $X \leqslant Z$ and $Y \geqslant Z$ imply that $Z = X$. Hence Z is in the same class as X and Y.

In the next paragraph, we give the necessary definitions that will help us obtain *all* most containing rewritings *efficiently*. Thus, Lemma 5.2 facilitates a pruning of all possible containment mappings in a similar fashion as in the

case without comparisons in the previous section. Also Examples 5.4 and 5.5 illustrate why the definitions in this paragraph are needed.

We are given a subgoal mapping together with a set E of exportable variables. Let $P(E)$ be a partition on a subset of view variables. We define $P(E)$ to be an *exporting subpartition* if it exports all variables in E (i.e., if we equate all variables in the same class, then each variable in E is equal to some distinguished variable). We define $P(E)$ to be a *maximal exporting subpartition* if there is no exporting subpartition that contains it. (As two exporting subpartitions of E may not refer to the same subset of view variables, we want to clarify what we mean by containment in such a setting: a subpartition $P(E)$ contains $P'(E)$ if each class of $P(E)$ is contained in a class of $P'(E)$.) Given a partition P_{S_0} on a set S_0 of variables and a subset S of S_0, we say that P_S is an induced exporting subpartition by a set E of variables if P_S exports E and each class of P_S is contained in a class of P_{S_0}. Given a subgoal mapping, any associated containment mapping induces (viewed as a partition on the set of view variables) an exporting subpartition on the set of exporting variables that the containment mapping uses.

Lemma 5.2. *All AC-containment mappings* (*viewed as partitions on the set of view variables*) *associated with*, (a) *a certain subgoal mapping*, (b) *a set of exporting variables*, *and* (c) *a maximal exporting subpartition form a semi-lattice*.

Proof. The proof is done along the lines of the proof of Lemma 4.1. We only need to additionally observe that by fixing a set of exporting variables E and a maximal exporting subpartition $P(E)$, any partition M_i of the view variables which exports the fixed set of variables E and induces the subpartition $P(E)$ has the properties of the partitions of containment mappings without comparisons. This means that the set of M_i's form a semi-lattice with respect to partition containment. □

The following lemma essentially says that it is sufficient for a certain subgoal mapping and set of exporting variables, to consider all partitions that induce one of the maximal subpartitions. That is, if we obtain all those associated rewritings, then all other rewritings are contained in them.

Lemma 5.3. *If $P(E)$ is a maximal subpartition, then there does not exist a partition on "all" view variables that exports E such that the induced subpartition by E properly contains $P(E)$.*

Proof. Towards contradiction, suppose the induced subpartition contains $P(E)$. Then $P(E)$ is not maximal. □

The following examples show why we also need to fix a set of exporting variables and a maximal exporting subpartition in the statement of the Lemma 5.2, i.e., they show that there are cases that we have more than one set of exporting variables, and cases where we have more than one maximal exporting subpartition.

Example 5.4. The first example shows a case where we have more than one set of exporting variables.

$$Q(X, Z) \text{:- } a(X, Y), a(Y, Z).$$

$$v(X, Z, A, B) \text{:- } a(X, Y), a(Y, Z), b(A, B), A \leqslant Y, Y \leqslant B.$$

There are two rewritings that correspond to the following two sets of exported variables: one is $\emptyset$, and the other one is $\{Y\}$.

$$P_1(X, Z) \text{:- } v(X, Z, A, B).$$

$$P_2(X, Z) \text{:- } v(X, Z_1, Y, Y), v(X_1, Z, Y, Y).$$

The expansion of $P_2(X, Z)$ is

$$P_2(X, Z) \text{:- } a(X, Y_1), a(Y_1, Z_1), b(Y, Y), Y \leqslant Y_1, Y_1 \leqslant Y, a(X_1, Y_2), a(Y_2, Z), Y_2 = Y$$

or

$$P_2(X, Z) \text{:- } a(X, Y), a(Y, Z_1), a(X_1, Y), a(Y, Z), b(Y, Y).$$

Note that P_1 and P_2 do not contain each other in either direction as queries. Also, note that the two rewritings occur because of the dual nature of the variable Y in v. Y can be treated as a nondistinguished variable, and that results in P_1. Y can also be treated as an exportable variable, that results in the second rewriting. The rewritings P_1 and P_2 do not relate to each other; hence we need to construct them both.

Example 5.5. We now give a second example for the "maximal exporting subpartition." Suppose we have the distinguished variables X_1, X_2, X_3, X_4, X_5 and the distinguishable variables Y_1, Y_2, Y_3 with the following ACs among them in the view. $X_1 \leqslant Y_1$, $X_1 \leqslant Y_3$, $X_2 \leqslant Y_1$, $X_2 \leqslant Y_2$, $Y_1 \leqslant X_3$, $Y_2 \leqslant X_4$, $Y_3 \leqslant X_5$. Suppose we want to export the variables Y_1, Y_2, and Y_3. Then there are the following two maximal exporting subpartitions:

- Subpartition 1: $\{X_1, X_5, Y_3\}, \{X_2, X_4, X_3, Y_1, Y_2\}$.
- Subpartition 2: $\{X_1, X_5, X_3, Y_1, Y_3\}, \{X_2, X_4, Y_2\}$.

Notice that there is no relation between them (i.e., no subpartition is a finer partition of the other); hence we need to consider them both in the algorithm.

Finally the above lemma leads to the main result:

Lemma 5.4. *Let M be a subgoal mapping with a set of exporting variables E and a maximal exporting subpartition $P(E)$. Let $\mathcal{P}$ be all the associated contained rewritings that export exactly E with subpartition $P(E)$. Then all rewritings in $\mathcal{P}$ form a semi-lattice with respect to query containment.*

The proof is a consequence of Lemma 5.2 and Proposition 5.3.

Corollay 5.1. *Given a total subgoal mapping with a set of exporting variables E and a maximal exporting subpartition $P(E)$, there exists an associated rewriting that contains all associated rewritings of this subgoal mapping that export exactly E with subpartition $P(E)$. We call this the most relaxed rewriting* (*containment mapping*, *respectively*).

5.2.3. Construction of legal MCDs

The same optimization can be applied as in the case without comparisons with some additional observation which concerns the ACs.

Lemma 5.5. *The elements of any maximal subpartition of a given set E are contained in the sets* leq-set *and* geq-set *of E formed by the inequality graph.*

Proof. Suppose P is a maximal exporting subpartition of E that uses a variable Y not in either of these sets. Then, by construction of these sets, for every variable X in E there is a variable u_X which is on a path (in the inequality graph) from Y to X. Hence u_X is in the same equivalence class as X, therefore by deleting Y, X is still exported. As this is true for any X in E, Y is redundant, hence P is not maximal; a contradiction. □

Lemma 5.6. *If we delete any element from leq-set or geq-set of E, there might exist a rewriting that is not contained in the contained rewriting generated by the algorithm.*

Proof. Easy to construct a counterexample. □

5.2.4. The algorithm

The algorithm contains the same three modules as the algorithm without arithmetic comparisons, which was presented in Section 4. Given a subgoal mapping, the procedure that finds the most relaxed associated containment mapping is the same where exported variables are treated as distinguished variables. The only difference is that the input also contains some a priori nonempty classes. Each of these classes contains variables that need to be equated for the exportable variables to be actually exported. The elements in these classes are found as explained in Section 5.2.2 by finding all maximal exporting subpartitions.

Before we give the algorithm that finds MCDs, we need to change the definition of a legal argument mapping as follows—the changes are marked by boldface. We say that an argument mapping is *legal* if the following is

Input: a CQAC query Q and a set V of CQAC views;
Output: a set of MCDs;
Method:

```
For each query subgoal g
    For each view v
        For each subgoal g' in v {
            Find legal argument mapping μ from g to g' (if there exists);
            Construct new partial MCD: (G = {g}, μ);
        }

While there are partial MCDs with shared variables {
    For each partial MCD (G, μ) {
        Choose shared variable X in G;
        If X is distinguishable {
            Mark X as exportable;
            Create new MCD with X being named as distinguished;
        }
        Consider as current the old MCD (G, μ);
        Choose query subgoal g that contains X;
        For each view subgoal g' that contains μ(X)
            If the extended subgoal mapping has a legal argument mapping and
            there exists an associated AC-containment mapping
                Replace the current partial MCD with new partial MCD: (G' = G ∪ {g}, μ' = ext(μ));
        }
    }

For each MCD {
    Consider the set S of exportable variables;
    For each maximal exporting subpartition of S
        construct a new MCD and add the necessary ACs (if doable) to obtain an AC-MCD.
}
return all AC-MCDs;
```

Fig. 6. Algorithm: finding MCDs.

true: (a) a distinguished variable is always mapped to a **either a distinguished or a distinguishable variable**, (b) whenever a constant is mapped to a constant, then it is the same constant, (c) whenever a constant is mapped to a variable, then this variable is **either a distinguished or a distinguishable variable**, (d) whenever a variable maps to a constant then it does not also map to a different constant, (e) two distinct constants do not map to the same variable.

We do not change the definition of shared variables, which we repeat here for convenience. We say that a *partial MCD has shared variables* if there is a variable X mapped to a nondistinguished view variable and there is a query subgoal in this partial MCD which contains X and X is shared with query subgoals that do not belong to this partial MCD. An MCD is defined to be a *minimal* partial MCD *without* shared variables (minimal w.r.t. the shared variable property, i.e., there is not a subset of the query subgoals and a subgoal mapping which is also an MCD) for which an associated containment mapping exists.

MCDs are also defined in the same way with the only difference that they include in their description a set of exported variables. However, we need to also define AC-MCDs, which are MCDs with a set of accompanying comparisons. In Fig. 6, we give the procedure that finds the MCDs.

The third procedure of the algorithm combines MCDs. We combine AC-MCDs in a similar way as before with the only difference that at the end we also check whether we need to add some arithmetic comparisons subgoals for the containment mapping from the query to the expansion to exist. To do that, we check whether the arithmetic comparisons in the expansion of the rewriting obtained from the definition of the view implies the associated ACs or whether the algorithm must add an AC to the rewriting explicitly. In the latter case, if the variable contained in the added AC is not

distinguished then we check whether there exists a variable Y in the geq-set (if the inequality is one of $<$ or $\leqslant$) or in the leq-set (if the inequality is one of $>$ or $\geqslant$) of the AC variable. We then add an (appropriate) inequality on Y to the rewriting. The following theorem proves that our algorithm is sound and complete.

Theorem 5.1. *Given a query and views that are CQACs for which the homomorphism property holds, the algorithm described above finds an MCR in the language of union of CQACs.*

Proof. The proof is similar to the corresponding theorem without comparisons. The proof for soundness is similar. The extra complications that are introduced by the ACs are apparent in the proof of completeness. This has been taken care of however in the proof of Lemma 5.4 whose direct consequence is the completeness of the algorithm. □

6. Recursive MCRs

In this section, we consider a wider class of queries than the class considered in the previous section. We allow for both LSI and RSI comparison subgoals in the query. In this case, we first argue that we cannot find an MCR unless we add some recursion in the language in which we express the rewritings. Then, we develop an algorithm which finds an MCR in the language of Datalog with arithmetic comparisons. In order to do so, however, we need to first find a query containment test that is easier than the general test in Theorem 2.1. It is also a contribution in query containment, because it finds another case where the containment problem is in NP. The structure of this section is as follows. Sections 6.1 and 6.2 discuss only query containment and obtain the result that simplifies the containment test in this case and also proves membership in NP in Theorem 6.2. The last subsection discusses rewritings and uses the result of Section 6.2 to develop an algorithm for finding MCRs. In more detail, we begin the section with an example that shows that we cannot find an MCR in the language of unions of CQACs and we observe that we might need recursion. Then, we restrict our attention to testing query containment in the special case where the containing query uses only one LSI subgoal or only one RSI subgoal (CQSI1). In Section 6.1, we argue using an example that checking for satisfaction of the containment entailment in this case is simpler, and then we prove some preliminary results. Section 6.2 proves that query containment in the case of an CQSI1 containing query can be reduced to containment of a CQ to a Datalog query. In the last subsection, we show how we use the result obtained in Section 6.2 to build an algorithm which constructs an MCR when given views which are CQSI and query is a CQSI1. We restrict attention to the case that only closed inequalities ($\leqslant$ and $\geqslant$) are used (i.e., no strict inequalities) because Theorem 2.2 simplifies the proofs.

Example 2.5 in Section 2 showed if some view variables are not distinguished, we can have an MCR that is a recursive Datalog program. The following example shows that if we only consider the language of finite unions of CQACs, the query Q does not have an MCR. This observation is not surprising given the results in [1], even though it does not follow directly from the results in that paper.

Example 6.1. Consider the following query and views:

$$Q\text{:- } e(X, Y), e(Y, Z), X \geqslant 5, Z \leqslant 8, red(Y).$$
$$v_1(X, Y)\text{:- } e(X, Y), X \geqslant 5, Y \leqslant 8.$$
$$v_2(X, Y)\text{:- } e(X, Z_1), e(Z_1, Z_2), e(Z_2, Z_3), e(Z_3, Y), red(X), red(Y), red(Z_2).$$

For each integer $k \geqslant 0$, we get a CR:

$$P_k\text{:- } v_1(X, Z_1), v_2(Z_1, Z_2), v_2(Z_2, Z_3), \ldots, v_2(Z_{k-1}, Z_k), v_1(Z_k, Y).$$

Proposition 6.1. *In Example* 6.1, *there is no finite union of CQACs which contains all P_ks and is contained in Q.*

Proof. Let there be a finite union of CQACs, R, that contains all P_k's and is contained in Q. Let s be the maximum number of subgoals in any rewriting $R_i \in R$. Consider P_k such that $k = s + 3$. Construct a view instance V by freezing the variables of the body of P_k to appropriate integers as follows: $v_1(X, Z_1)$ is frozen to $v_1(6, 4)$ and $v_1(Z_k, Y)$ is frozen to $v_1(9, 3)$ and, the rest is frozen to any distinct integers. Clearly P_k is true on V. Since R contains P_k, there exists a rewriting $R_i \in R$ of size less than or equal to s that is true on V. R_i uses at most s tuples in V to satisfy its body and produce a valid head. Produce a view instance V' that contains only the s tuples used to produce a valid head for R_i. Since V' contains s tuples, whereas V contained $s + 3$, at least one $v_2(Z_j, Z_{j+1})$ tuple that was in V is not present in

V'. Now, construct a database D' from V' by replacing the tuples in V' with their expansions. For example, we replace the first tuple $v_1(6, 4)$ by the tuple $e(6, 4)$, the last tuple $v_1(9, 3)$ by the tuple $e(9, 3)$, and so on. Replace the variables in the expansion of $v_2(Z_j, Z_{j+1})$ with distinct values $\geqslant 8$, for $l \leqslant j$, and with distinct values $\leqslant 5$, for $l \geqslant j+1$; e.g., replace $v_2(a, b)$ by $e(a, 16)$, $e(16, 17)$, $e(17, 18)$, $e(18, b)$ if a, b are the frozen counterparts of variables Z_{j-4}, Z_{j-3}. Query Q is not true on D' because it requires a red integer in the middle of two consecutive e relations with ends being $\geqslant 5$ (the starting end) and $\leqslant 8$ (the other end). However, as R_i is contained in Q, Q is true on D',—a contradiction. Therefore, there exists no finite union of CQACs that contains all P_ks and is contained in Q. □

6.1. CQAC-SI containment: preliminaries

The following is a motivating example showing that testing containment for CQAC-SI queries can be somewhat simplified compared to the general case.

Example 6.2. Consider the following two queries:

$$Q_1()\text{:- } e(X, Y), e(Y, Z), X \geqslant 5, Z \leqslant 8$$
$$Q_2()\text{:- } e(A, B), e(B, C), e(C, D), e(D, E), A \geqslant 6, E \leqslant 7.$$

There are three containment mappings from the ordinary subgoals of Q_1 to the ordinary subgoals of Q_2:

$$\mu_1\text{: } X \to A, Y \to B, Z \to C,$$
$$\mu_2\text{: } X \to B, Y \to C, Z \to D,$$
$$\mu_3\text{: } X \to C, Y \to D, Z \to E.$$

The following entailment holds:

$$A \geqslant 6 \wedge E \leqslant 7 \Rightarrow \mu_1(X \geqslant 5 \wedge Z \leqslant 8) \vee \mu_3(X \geqslant 5 \wedge Z \leqslant 8).$$

Hence, by Theorem 2.2, Q_2 is contained in Q_1.[2]

Now we want to examine in more detail a proof that shows this entailment to be true. For this purpose, let us rewrite it as

$$A \geqslant 6 \wedge E \leqslant 7 \Rightarrow (A \geqslant 5 \wedge C \leqslant 8) \vee (C \geqslant 5 \wedge E \leqslant 8).$$

It is equivalent to

$$A \geqslant 6 \wedge E \leqslant 7 \Rightarrow (A \geqslant 5 \vee C \geqslant 5) \wedge (A \geqslant 5 \vee E \leqslant 8) \wedge (C \leqslant 8 \vee C \geqslant 5) \wedge (C \leqslant 8 \vee E \leqslant 8).$$

The latter holds because

1. $A \geqslant 6 \Rightarrow A \geqslant 5$, and $E \leqslant 7 \Rightarrow E \leqslant 8$.
2. $true \Rightarrow C \leqslant 8 \vee C \geqslant 5$.

In other words, the entailment of each conjunct in the right-hand side follows from one of the two following reasons:

1. because a single inequality in the left-hand side implies a single inequality in the right-hand side (called a *direct implication*);
2. because the disjunction of two inequalities in the right-hand side is true (called *coupling implication*).

It turns out that this observation can be generalized even in the case the left-hand side contains any arithmetic comparisons. In the following lemma, we prove that whenever we want to derive a disjunction of SI inequalities from a given set of inequalities, we only need to consider these two kinds of implications.

Lemma 6.1. (1) *Let $b_1, \ldots, b_k$ be the closure*[3] *of a set of inequalities and $e_1, \ldots, e_n$ be SI inequalities. Then*

$$b_1 \wedge \cdots \wedge b_k \Rightarrow e_1 \vee \cdots \vee e_n$$

---

[2] Remember that $\mu_1(X \geqslant 5 \wedge Z \leqslant 8)$ denotes $\mu_1(X) \geqslant 5 \wedge \mu_1(Z) \leqslant 8$ which, under the given mapping μ_1 is equivalent to $A \geqslant 5 \wedge C \leqslant 8$. Similarly for any μ_i.

[3] The closure of a set S of inequalities contains all inequalities implied by the conjunction of the inequalities in S.

iff either (*a*) *there are* b_k *and* e_i *such that* $b_k \Rightarrow e_i$ (direct implication), *or* (*b*) *there are* e_i *and* e_j *and* $b_k = X\,\theta\,Y$ *such that* $X\,\theta\,Y \Rightarrow e_i \vee e_j$ (θ-coupling implication) *or* (*c*) *there are* e_i *and* e_j *such that* $true \Rightarrow e_i \vee e_j$ (coupling implication).

(2) *Let* $b_1, \ldots, b_k$ *and* $e_1, \ldots, e_n$ *be SI inequalities. Then*

$$b_1 \wedge \cdots \wedge b_k \Rightarrow e_1 \vee \cdots \vee e_n$$

iff either (*a*) *there are* b_k *and* e_i *such that* $b_k \Rightarrow e_i$ (direct implication), *or* (*b*) *there are* e_i *and* e_j *such that* $true \Rightarrow e_i \vee e_j$ (coupling implication).

Proof. Observe that

$$b_1 \wedge \cdots \wedge b_k \Rightarrow e_1 \vee \cdots \vee e_n$$

is equivalent to

$$\neg(b_1 \wedge \cdots \wedge b_k) \vee e_1 \vee \cdots \vee e_n$$

which is equivalent to

$$\neg(b_1 \wedge \cdots \wedge b_k \wedge \neg e_1 \wedge \cdots \wedge \neg e_n)$$

which is equivalent to

$$b_1 \wedge \cdots \wedge b_k \wedge \neg e_1 \wedge \cdots \wedge \neg e_n \Rightarrow false.$$

We can easily prove that the last implication holds iff there is a cycle in the inequality graph of the inequalities $b_1, \ldots, b_k, \neg e_1, \ldots, \neg e_n$, which contains at least one edge with label being a strict inequality.

The result now is an immediate consequence of the fact that cycles that contain SI inequalities are only of these two (three, respectively) kinds (see also [36, p. 886] for a complete set of inference rules that derive all inequalities implied from a given set of inequalities). □

Now we focus on entailments that have the pattern of the entailment asked to be proven in the CQAC containment test of Theorem 2.2, that is, on the left-hand side of the entailment we have the closure of a set of inequalities and on the right-hand side we have a disjunction where each disjunct is a conjunction of inequalities. For ease of reference, we call these entailments *containment entailments* (although it is not necessary that they have to relate to a query containment test). Moreover, we have the following constraints: (a) the inequalities used in the right-hand side are only SI inequalities and (b) in each disjunct in the right-hand side there are a number of LSI (RSI, respectively) inequalities and at most one RSI (LSI, respectively) inequality. We call these *SI1 containment entailments*.

The following lemma is an easy observation.

Lemma 6.2. *Let* $\mathcal{E}$ *be an SI1 containment entailment. Then there is at least one disjunct* d_i *for which the following holds: there is at most one inequality in* d_i *that is not directly implied by the left-hand side. We call* d_i *a* leaf disjunct.

Proof. We prove by contradiction. Suppose there is no leaf disjunct. Then each disjunct contains at least two inequalities that are not directly implied by the left-hand side. Since each disjunct contains at most one RSI (LSI, respectively), there is no disjunct that contains two RSI (LSI, respectively) inequalities that are not directly implied by the left-hand side. Hence, the following claim: All disjuncts contain at least one LSI (RSI, respectively) which is not directly implied by the left-hand side.

Applying distributive law, we can turn equivalently the right-hand side of the entailment into a conjunction. Based on the above claim, we deduce that there is a conjunct which contains only LSI (RSI, respectively) inequalities each of which is not directly implied by the left-hand side. However, according to Lemma 6.1 the only other choice for the entailment to be satisfied is for a coupling inequality to hold. But this is impossible when we have only LSI or only RSI inequalities. Hence, this entailment is not true, contradiction. □

Finally, the technical lemma that follows is one of the main technical tools used in the proof in the next subsection.

Lemma 6.3. *Let $\mathcal{E}$: $\beta \Rightarrow \beta_{a_1} \vee \beta_{a_2} \vee \cdots \vee \beta_{a_k}$ be a SI1 containment entailment that contains more than one disjunct. Suppose that $\mathcal{E}$ is true and also, if we drop any of the disjuncts then $\mathcal{E}$ does not hold. Let $\mathcal{E}$ contain k disjuncts and let β_{a_i} be a leaf disjunct and let e be the inequality in β_{a_i} that is not directly implied by β (see Lemma* 6.2).

Then the following SI1 containment entailment $\mathcal{E}'$ that has $k-1$ disjuncts is also true: (suppose wlog that $\beta_{a_i} = \beta_{a_k}$):

$$\beta \wedge \neg e \Rightarrow \beta_{a_1} \vee \beta_{a_2} \vee \cdots \vee \beta_{a_{k-1}}$$

Proof. We deduce from

$$\beta \Rightarrow \beta_{a_1} \vee \beta_{a_2} \vee \cdots \vee \beta_{a_k}$$

that

$$\beta \wedge \neg \beta_{a_k} \Rightarrow \beta_{a_1} \vee \beta_{a_2} \vee \cdots \vee \beta_{a_{k-1}}$$

or equivalently (assuming $\beta_{a_k} = e_1 \wedge \cdots \wedge e_t$, where e_is are single inequalities)

$$(\beta \wedge \neg e_1) \vee (\beta \wedge \neg e_2) \vee \cdots \vee (\beta \wedge \neg e_t) \Rightarrow \beta_{a_1} \vee \beta_{a_2} \vee \cdots \vee \beta_{a_{k-1}}.$$

Assume wlog that $e = e_1$. Since each e_i except e_1 is entailed by β, each disjunct except the first one in the lhs is always false. Hence, the latter entailment is equivalent to

$$\beta \wedge \neg e \Rightarrow \beta_{a_1} \vee \beta_{a_2} \vee \cdots \vee \beta_{a_{k-1}}. \qquad \square$$

6.2. CQCA-SI containment: a reduction

In this subsection, we want to check containment of CQAC queries in the case the containing query uses only SI inequalities and \ it either uses a single LSI inequality or a single RSI inequality. We call them CQAC-SI1 queries.

First we show how to reduce containment in this case to containment of a CQ to a Datalog query. The reduction is done as follows: suppose we want to check whether Q_1 contains Q_2. We will first transform Q_2 into a CQ Q_2^{CQ} and Q_1 into a Datalog query Q_1^{Datalog} and then we will prove that checking containment of Q_2 in Q_1 is equivalent to checking containment of Q_2^{CQ} in Q_1^{Datalog}. Without loss of generality, we restrict attention in this section to boolean queries.

We will describe the construction of Q_1^{Datalog}, Q_2^{CQ} in parallel with an example.

Construction of Q_2^{CQ}: We introduce new unary EDBs [36], two for each constant c in Q_2, namely $U_{\geqslant c}$ and $U_{\leqslant c}$. For each AC of the form $X\theta c$, we refer to $U_{\theta c}$ as the *associated U*-predicate.

One rule for Q_2^{CQ}: We copy the regular subgoals of Q_2 and for each AC predicate $X_i\theta c_i$ in β_2 we add a unary predicate subgoal $U_{\theta c_i}(X_i)$.

Example 6.3. *Consider two queries:*

$$Q_1 \text{:- } e(X, Y), e(Y, Z), X \geqslant 5, Z \leqslant 8.$$
$$Q_2 \text{:- } e(A, B), e(B, C), e(C, D), e(D, E), A \geqslant 6, E \leqslant 7.$$

Q_1 contains Q_2. For Q_2, we construct Q_2^{CQ}.

$$Q_2^{\text{CQ}} \text{:- } e(A, B), e(B, C), e(C, D), e(D, E), U_{\geqslant 6}(A), U_{\leqslant 7}(E).$$

Construction of Q_1^{Datalog}: We construct three kinds of rules, *mapping rules, coupling rules* and *link rules*. Also, we construct a single *query rule*.

We introduce new unary IDBs [36], two pairs for each constant c in Q_1, namely $I_{\geqslant c}$, $I_{\leqslant c}$ and $J_{\geqslant c}$, $J_{\leqslant c}$. We also use all unary EDB predicates we introduced for Q_2^{CQ} in the link rules. For each pair of one inequality $X\theta c$ and one IDB predicate atom $I_{\theta c}(X)$ ($J_{\theta c}(X)$, respectively), we refer to each other as the *associated I-atom* (*associated J-atom*, respectively) or the *associated AC*.

The query rule copies in its body all subgoals of Q_1 and replaces each AC subgoal of Q_1 by its associated I-atom. We get one mapping rule for each single inequality, e, in Q_1. The body is a copy of the body of the query rule, only that the I atom associated to e is deleted. The head is the J atom associated to e.

For every pair of constants $c_1 \leqslant c_2$ used in Q_1, we construct two coupling rules. One rule is $I_{\leqslant c_2}(X)$:- $J_{\geqslant c_1}(X)$ and the other is $I_{\geqslant c_1}(X)$:- $J_{\leqslant c_2}(X)$.

Finally, we construct the link rules: for each pair of constants (c_1, c_2) from Q_1, Q_2, respectively, if $X\,\theta\,c_2$ entails $X\,\theta\,c_1$, we construct the rule: $I_{\theta c_1}(X)$:- $U_{\theta c_2}(X)$.[4]

Example 6.4. We continue on the previous example. For Q_1, we construct a Datalog program Q_1^{Datalog}:

Q_1^{Datalog}:- $e(X, Y), e(Y, Z), I_{\geqslant 5}(X), I_{\leqslant 8}(Z)$ query rule,
$J_{\leqslant 8}(Z)$:- $e(X, Y), e(Y, Z), I_{\geqslant 5}(X)$ mapping rule,
$J_{\geqslant 5}(X)$:- $e(X, Y), e(Y, Z), I_{\leqslant 8}(Z)$ mapping rule,
$I_{\leqslant 8}(X)$:- $J_{\geqslant 5}(X)$ coupling rule,
$I_{\geqslant 5}(X)$:- $J_{\leqslant 8}(X)$ coupling rule,
$I_{\geqslant 5}(X)$:- $U_{\geqslant 6}(X)$ link rule,
$I_{\leqslant 8}(X)$:- $U_{\leqslant 7}(X)$ link rule.

The two last rules are *link* rules, and they will change if we change the query Q_2. The other rules depend only on Q_1.

The intuition as to the reason this construction is expected to work is as follows. The unary predicates (both IDBs and EDBs) in the Datalog program are used to mark whether the argument of the predicate satisfies an inequality of the form $X\,\theta\,c$ (c is a constant) (the subscript in the predicate name is a reminder of which inequality). Actually the J predicates are used as reminders that a coupling inequality is needed whereas the I and U predicates are used in the role of either "direct" implication or that the coupling inequality is provided.

Each link rule encodes an entailment of the form $X \leqslant 7 \Rightarrow X \leqslant 8$, i.e., it encodes in general an entailment $X \leqslant c_1 \Rightarrow X \leqslant c_2$, where $c_1 \leqslant c_2$. A coupling rule is motivated by part 2(b) of Lemma 6.1. A mapping rule encodes a mapping from Q_1 to Q_2. Lemmas 6.2 and 6.3 provide the support for all the technical details to go through. Now note that any CQ Q_π produced by the Datalog program[5] can be viewed as the union of copies of the subgoals of Q_1. Thus, a mapping from Q_π into the subgoals of Q_2 can be thought of as a set of mappings from the ordinary subgoals of Q_1 into the ordinary subgoals of Q_2.

Thus, according to our claim, in our running example we expect that the conjunctive query Q_2^{CQ} produced by the transformation is contained in the Datalog query Q_1^{Datalog}. This is easy to see, however we show the details in the example that follows.

Example 6.5. We continue on the previous example. To show that Q_1^{Datalog} contains Q_2^{CQ}: unfold rule 5 into the query rule:

$$Q_1^{\text{Datalog}}\text{:- } e(X, Y), e(Y, Z), J_{\leqslant 8}(X), I_{\leqslant 8}(Z).$$

Unfold rules 2 and 3 into the above and get

$$Q_1^{\text{Datalog}}\text{:- } e(X, Y), e(Y, Z), e(X1, Y1), e(Y1, X), I_{\geqslant 5}(X1), I_{\leqslant 8}(Z).$$

Unfolding the four last rules into it, we get

$$Q_1^{\text{Datalog}}\text{:- } e(X, Y), e(Y, Z), e(X1, Y1), e(Y1, X), U_{\geqslant 6}(X1), U_{\leqslant 7}(Z).$$

The latter is a CQ produced by the Datalog program, and this CQ maps on Q_2^{CQ}, thus showing the containment.

---

[4] The link rules are the only rules of Q_1^{Datalog} that depend on Q_2; actually they relate the comparison predicates of Q_1 to the comparison predicates of Q_2.

[5] A Datalog program is equivalent to the union of all CQs produced by unfolding the rules several times until no recursive predicates are contained.

The following theorem is the main technical result of this section.

Theorem 6.1. *Let Q_1 be a CQAC-SI1 query and Q_2 be a CQAC-SI query. Let Q_1^{Datalog} be the transformed Datalog query of Q_1 and Q_2^{CQ} be the transformed CQ of Q_2. Then Q_1 contains Q_2 iff Q_1^{Datalog} contains Q_2^{CQ}.*

Proof. Suppose Q_1:- $Q_{10} + \beta_1$ and Q_2:- $Q_{20} + \beta_2$. The "if" direction. Since Q_1^{Datalog} contains Q_2^{CQ}, there is a computation $\mathcal{C}$ of Q_1^{Datalog} on the canonical database of Q_2^{CQ} which returns the answer "yes" to the boolean query. Following this computation, we will construct, a set of mappings $\mu_1, \mu_2, \ldots, \mu_n$ from Q_{10} to Q_{20} which will satisfy

$$\beta_2 \Rightarrow \mu_1(\beta_1) \vee \mu_2(\beta_1) \vee \cdots \vee \mu_n(\beta_1).$$

Computation $\mathcal{C}$ consist a number of stages, each stage consisting of an application of a mapping rule of Q_1^{Datalog}. (Between stages, there might be a number of coupling rules fired but this counts still for one stage. Link rules are fired only in the leaves of the computation.) We construct one mapping for each stage, i.e., one mapping for each application of a mapping rule.

The proof is done by induction on the number of stages required for a ground fact to be added in a J-predicate relation.

Inductive hypothesis: Suppose that the J-atom $J_{\theta c}(x)$ is computed at stage $\leqslant l$. Let $\mu_1, \mu_2, \ldots, \mu_l$ be all the mappings used for firing the mapping rules. Then, it holds

$$\beta_2 \Rightarrow \mu_1(\beta_1) \vee \mu_2(\beta_1) \vee \cdots \vee \mu_l(\beta_1) \vee \neg(x\theta c).$$

Proof of the induction: The basis step is easy. For the inductive step, suppose fact $J_{\theta c}(x)$ was computed at stage k. In the top of the computation tree of this fact, a mapping rule is used. In order to fire this mapping rule, we used some I-facts. Those I-facts are computed from J-facts using coupling rules. Naturally those J-facts were computed at stages $\leqslant k$. Suppose that these J-facts are $J_{\theta_i c_i}(x_i)$, $i = 1, \ldots,$ and each is computed using a set of mappings $S_i = \{\mu_1^i, \ldots, \mu_{l_i}^i\}$.

Assume that $\mu_{\text{new}}(\beta_1) = e_1 \vee \cdots \vee e_t$, where e_js are single inequalities. By construction, for each set S_i there is a j_i such that $\neg(x_i\theta_i c_i) \Rightarrow e_{j_i}$. This covers all e_js except one, the one associated to $J_{\theta c}(x)$, suppose this is the e_t. Then, for each S_i, we get

$$\beta_2 \Rightarrow \mu_1^i(\beta_1) \vee \mu_2^i(\beta_1) \vee \cdots \vee \mu_{l_i}^i(\beta_1) \vee e_{j_i}$$

or equivalently,

$$\beta_2 \Rightarrow \mu_1^i(\beta_1) \vee \mu_2^i(\beta_1) \vee \cdots \vee \mu_{l_i}^i(\beta_1) \vee e_{j_i} \vee \neg e_t$$

Since, for the not covered e_t, we can also write: $\beta_2 \Rightarrow e_t \vee \neg e_t$, we end up with the desired entailment:

$$\beta_2 \Rightarrow \neg(x\theta c) \vee \mu_{\text{new}}(\beta_1) \vee \bigvee_{all\ \ S_i} \mu_1^i(\beta_1) \vee \mu_2^i(\beta_1) \vee \cdots \vee \mu_{l_i}^i(\beta_1).$$

The "only if" direction. Since Q_1 contains Q_2, there are mappings $\mu_1, \mu_2, \ldots, \mu_n$ from the regular subgoals of Q_1 to the regular subgoals of Q_2 such that: $\beta_2 \Rightarrow \mu_1(\beta_1) \vee \mu_2(\beta_1) \vee \cdots \vee \mu_n(\beta_1)$. Intuitively, we will prove this direction, by proving that the mappings $\mu_1, \mu_2, \ldots, \mu_n$ provide all the mappings that will fire the mapping rules in the computation of Q_1^{Datalog} on the canonical database of Q_2^{CQ}. Recall that, by construction, the canonical database of Q_2^{CQ} contains the frozen ordinary subgoals of Q_2 and all the U facts associated with inequalities in β_2.

We prove the general case of this direction by induction on the number n of mappings.

Inductive hypothesis: For all $n \leqslant k$ it holds: Let $\mu_1, \mu_2, \ldots, \mu_n$ be mappings from the ordinary subgoals of Q_1 to the ordinary subgoals of Q_2 and e_i, $i = 1, \ldots, L$ be an SI inequality from β_1 with its variable replaced by a variable of Q_2. Suppose that the following entailment holds: $\mathcal{E}:\ \ \beta_2 \vee \neg e_1 \vee \cdots \vee \neg e_L \Rightarrow \mu_1(\beta_1) \vee \mu_2(\beta_1) \vee \cdots \vee \mu_n(\beta_1)$. Consider the Datalog query Q_1^{Datalog} applied on the union of the canonical database of Q_2^{CQ} and the set of the following facts (on elements of the domain of the canonical database of Q_2^{CQ}): a fact $I_{\theta c}(x)$ is added for each $e_i = X\theta c$ in $\mathcal{E}$ (x is the frozen variable for X in the canonical database). Then the answer that Q_1^{Datalog} returns is "yes".

Proof of the initial step ($k = 1$): We have two cases: (a) If there are no e_i's in $\mathcal{E}$ then $\mathcal{E}: \ \beta_2 \Rightarrow \mu_1(\beta_1)$. Because of Lemma 6.1 this implication is true only if there is a direct implication for every inequality in the right-hand side of $\mathcal{E}$. All direct implications however are captured in the link rules of the Datalog query which therefore are fired and I facts are produced which together with the mapping μ_1 fire the query rule. b) There are e_i's in $\mathcal{E}$. The argument is the same only that now the direct implications are not all captured by the link rules, hence some I facts may not be produced by the link rules. These facts however are added to the database by construction (see inductive hypothesis), hence, the query rule is again fired.

Proof of the inductive step: Given an entailment $\mathcal{E}: \ \beta_2 \vee \neg e_1 \vee \cdots \vee \neg e_L \Rightarrow \mu_1(\beta_1) \vee \mu_2(\beta_1) \vee \cdots \vee \mu_{k+1}(\beta_1)$ with $k+1$ disjuncts, according to Lemma 6.3 the following entailment is also true: $\mathcal{E}': \ \beta_2 \vee \neg e_1 \vee \cdots \vee \neg e_L \vee \neg e_{\text{new}} \Rightarrow \mu_1(\beta_1) \vee \mu_2(\beta_1) \vee \cdots \vee \mu_k(\beta_1)$. According to the inductive hypothesis, the Datalog query answers "yes" on the canonical database D of Q_2^{CQ} with the I facts associated to $e_1, \ldots, \neg e_L, \neg e_{\text{new}}$ added. To prove the inductive step, we need to prove that the Datalog query applied on database D after we remove the I fact for e_{new} answers "yes" too. This is true because, the removed fact is added by the application of a mapping rule and a coupling rule: The mapping rule uses the mapping μ_{k+1} and produces a new J fact and a coupling rule produces the deleted I fact from this J fact. □

Proposition 6.2. *The reduction described above is polynomial.*

Proof. For the containing query: we have only one query rule of size linear on the size of one of the queries and we have one mapping rule for each comparison subgoal of size again linear. We have a number of coupling rules and link rules of constant size each and their number is at most quadratic on the number of comparison sublgoals. □

The following result is a consequence of this reduction.

Theorem 6.2. *The problem of testing whether a CQSI query is contained in a CQSI1 query is in NP.*

Proof. The reduction described in this section is a polynomial reduction. Also the Datalog program that we are constructing is *monadic*, i.e., all its IDB predicates are of arity less or equal to 1. Thus, it suffices to show that testing whether a CQ Q_2 is contained in a monadic Datalog query Q_1 is in NP. (In the general case, this problem is EXPTIME-complete.)

For the special case of monadic Datalog (wlog assume boolean queries), we argue as follows: the test is to run the Datalog query Q_1 on the canonical database D_2 of the CQ Q_2. Q_2 is contained in Q_1 iff it returns the answer "yes."

The certificate is: (a) the unary IDB facts computed (polynomially many), (b) the derivation tree that computes them (polynomial in size, if we do not repeat nodes–instead redirect the links. That is, we describe the tree using an acyclic graph), (c) for each fact the mapping from the subgoals of Q_1 to the subgoals of Q_2 which computes this fact.

Test that the certificate proves that the answer is "yes": (a) Test that the derivation tree is a tree or equivalently that its succinct description is a directed acyclic graph. (b) Test that the given containment mappings are using only IDB facts that are children (in the derivation tree) of the currently computed IDB fact. (c) Test that each of the mappings is a containment mapping.

6.3. Finding MCR

We use the result in the previous section to construct an algorithm that produces an MCR given a CQAC-SI1 query and CQAC-SI views. Our algorithm reduces this problem to the problem of finding an MCR given a Datalog query and conjunctive views (without arithmetic comparisons) and then uses the algorithm in [18].

We need the following lemma. It says that we do not need to consider contained rewritings that use other arithmetic comparisons besides semi-interval.

Lemma 6.4. *Let query Q and views V be CQSI. Let P be a contained rewriting of Q using views V. Then, there is a finite union of contained rewriting of Q using V, $P_1, \ldots, P_k$, which contains P and uses only SI ACs.*

Proof. Let $P = P_0 + \beta_P = P_0 + \beta_P^{\text{SI}} + \beta_P^{\text{rest}}$, where β_P^{SI} are the SI comparisons of β_P and β_P^{rest} are the remaining comparisons of β_P. We construct $P_1, \ldots, P_k$ as follows: the head of P_i is the same as the head of P. The body of

P_i contains a copy of all ordinary subgoals of P, all SI inequalities in the closure of β_P and some additional SI ACs. These additional SI ACs cover all possible placements of the variables of P with respect to the constants in Q that are consistent with the inequalities in β_P. In particular, for constants $c1 \leqslant c2 \leqslant c3 \leqslant \ldots \leqslant cm$, we consider 2m+1 intervals: $(-\infty, c1](c1, c2], \ldots..$ Thus, $P_i' = P_0 + \beta_P^{\mathrm{SI}} + \beta_i$, where β_i contains only SI inequalities and define a specific placement of the variables in P_0 w.r.t. all constants in Q. For an example, suppose that we have two constants used in the query and views, say 5 and 15 and we have two variables X, Y in the rewriting P. Then we have nine different ways to place the variables in the intervals $(-\infty, 5]$, $(5, 15]$, $(15, \infty)$, thus we form nine new rewritings. One of these rewritings, e.g., is P with the following β_i added: $\beta_i = X \leqslant 5, Y \leqslant 5$.

It is easy to see that the union of those rewritings contain P: We may think of the P_is as follows: we can rewrite P equivalently as a union of contained rewritings, $P_1', \ldots, P_k'$. The body of each P_i' is the same as the body of P. P_i' has some additional SI ACs, the ones in β_i. Clearly the union of P_i's contains P. Now each P_i that we constructed is actually P_i' with some ACs dropped. But this is more containing than P_i', hence the union of P_is contains P too.

It remains to be proven that each of them is a contained rewriting in the query. Let $\beta_P = \beta_P^{\mathrm{SI}} + \beta_P^{\mathrm{rest}}$. We prove that $P_i = P_0 + \beta_P^{\mathrm{SI}} + \beta_i$ is still contained in Q. We consider the expansions of P and P_i. Let $P^{\mathrm{exp}} = P_0^{\mathrm{exp}} + \beta_P^{\mathrm{SI}} + \beta_P^{\mathrm{rest}} + \beta_{\mathrm{views}}$ and let $P_i^{\mathrm{exp}} = P_0^{\mathrm{exp}} + \beta_P^{\mathrm{SI}} + \beta_i + \beta_{\mathrm{views}}$. We will prove that if P^{exp} is contained in Q then P_i^{exp} is contained in Q too. The proof is based on the following claim which is an easy consequence of Lemma 6.1.

Claim. *Let $\mathcal{E}$ be a containment entailment that contains only SI inequalities in the right-hand side. Turn the right-hand side of $\mathcal{E}$ into a conjunction of disjunctions. Then $\mathcal{E}$ holds iff the following is true*: *For each conjunct, one of the three conditions in Lemma* 6.1.1 *holds*.

The entailment $\mathcal{E}$ that proves containment of P^{exp} in Q differs from the entailment $\mathcal{E}_i$ that proves containment of P_i^{exp} in Q in the following: the left-hand side of $\mathcal{E}$ may contain some ACs of the form $X\,\theta\,Y$ that are not contained in $\mathcal{E}_i$. The only condition in Lemma 6.1.1 which uses such inequalities is the θ-coupling condition. So it suffices to argue on this condition.

Suppose that one of the conditions that prove that $\mathcal{E}$ holds is: $X \leqslant Y \Rightarrow X \geqslant c1 \vee Y \leqslant c2$. Then, the only P_i whose SIs do not entail $X \geqslant c1 \vee Y \leqslant c2$ is the one which contains the SIs $X \leqslant c1 \wedge Y \geqslant c2$. But this is inconsistent with $X \leqslant Y$, hence this P_i was discarded during the construction. □

Algorithm:

1. For the query Q, we construct the Datalog query Q^{Datalog}. We use the construction in the previous section.
2. For each view v_i, we construct a new view v_i^{CQ}. We use the construction in the previous section.
3. We also construct a new set of views, $u_{\theta c}$, one for each unary predicate $U_{\theta c}$. The definition is $u_{\theta c}(X)$:- $U_{\theta c}(X)$.
4. We find an MCR P for the Datalog query Q^{Datalog} using the views v_i^{CQ}'s and $u_{\theta c}$'s [18].
5. To obtain an MCR P_0 for Q, we replace in the found MCR P each v_i^{CQ} by v_i and each $u_{\theta c}(X)$ by AC $X\theta c$.

The correctness of the algorithm is based on the following proposition.

Proposition 6.3. *Let Q and V be CQAC-SI and Q be CQAC-SI1 and let Q^{Datalog} and the views V' be as in the algorithm. Let P, P_0 be as in the algorithm. Then P is an MCR of Q^{Datalog} using V' iff P_0 is an MCR of Q using V.*

Proof. The proof is based on Lemma 6.4 and Theorem 6.1. According to Lemma 6.4, any (possibly infinite) union of contained rewritings (that are CQACs) in Q is contained in a (possibly infinite) union of contained rewritings in Q that use only SI comparisons. Hence, if an MCR exists in the language of (possibly infinite) union of CQACs then an MCR exists in the language of (possibly infinite) union of CQAC-SIs. Each CQAC-SI P_i has an expansion P_i^{exp} that is contained in Q. According to Theorem 6.1 this is equivalent to the following: $P_i^{\mathrm{exp-CQ}}$ (that is the transformed P_i^{exp} as in Theorem 6.1) is contained in Q^{Datalog}. However, $P_i^{\mathrm{exp-CQ}}$ can be viewed also as the expansion of a rewriting P_i^{CQ} of Q^{Datalog} using V', where P_i^{CQ} is P_i with views from V replaced by views from V' and unary EDBs $U_{\theta c}(X)$ replaced by comparisons $X\theta c$. □

The following theorem proves correctness of the algorithm and is a straightforward consequence of the above proposition.

Theorem 6.3. *Given a query Q which is CQAC-SI1 and views V which are CQAC-SI, the algorithm finds an MCR of Q using V.*

7. Future work and conclusion

We believe that the problem of answering queries using views in the presence of arithmetic comparisons is fundamental to any database system using views. This paper identifies cases where the problem can be solved and provides algorithms to do so. Specifically, we have developed an efficient algorithm to obtain MCRs for LSI queries. We have also shown that recursive datalog programs are necessary to rewrite semi-interval queries and identified subcases where there is an MCR in datalog with comparisons and provided an algorithm to find it.

The decidability of finding an MCR of a query with comparison predicates using views with comparison predicates, especially, when all the view variables are not distinguished, needs to be investigated for other subcases too.

Acknowledgments

We thank Jeff Ullman for many useful suggestions and also for the proof of Theorem 3.1.

References

[1] S. Abiteboul, O.M. Duschka, Complexity of answering queries using materialized views, in: PODS, 1998, pp. 254–263.
[2] F. Afrati, R. Chirkova, M. Gergatsoulis, V. Pavlaki, Finding equivalent rewritings in the presence of arithmetic comparisons, in: EDBT, 2006.
[3] F.N. Afrati, M. Gergatsoulis, T.G. Kavalieros, Answering queries using materialized views with disjunctions, in: ICDT, 1999, pp. 435–452.
[4] F. Afrati, C. Li, P. Mitra, Answering queries using views with arithmetic comparisons, in: PODS, 2002.
[5] F. Afrati, C. Li, P. Mitra, On containment of conjunctive queries with arithmetic comparisons, in: EDBT, 2004.
[6] F. Afrati, C. Li, J.D. Ullman, Generating efficient plans using views, in: SIGMOD, 2001, pp. 319–330.
[7] R.J. Bayardo Jr., et al., Infosleuth: semantic integration of information in open and dynamic environments (experience paper), in: SIGMOD, 1997, pp. 195–206.
[8] C. Beeri, A.Y. Levy, M.-C. Rousset, Rewriting queries using views in description logics, in: PODS, ACM Press, New York, July-August 1997, pp. 99–108.
[9] D. Calvanese, G. De Giacomo, M. Lenzerini, Answering queries using views over description logics knowledge bases, in: PODS, 2000, pp. 386–391.
[10] A.K. Chandra, H.R. Lewis, J.A. Makowsky, Embedded implication dependencies and their inference problem, in: STOC, 1981, pp. 342–354.
[11] A.K. Chandra, P.M. Merlin, Optimal implementation of conjunctive queries in relational data bases, in: STOC, 1977, pp. 77–90.
[12] S. Chaudhuri, R. Krishnamurthy, S. Potamianos, K. Shim, Optimizing queries with materialized views, in: ICDE, 1995, pp. 190–200.
[13] S. Chaudhuri, M.Y. Vardi, On the equivalence of recursive and nonrecursive datalog programs, in: PODS, 1992, pp. 55–66.
[14] S.S. Chawathe, et al., The TSIMMIS project: integration of heterogeneous information sources, in: IPSJ, 1994, pp. 7–18.
[15] JC. Chekuri, A. Rajaraman, Conjunctive query containment revisited, in: F.N. Afrati, Ph.G. Kolaitis (Eds.), ICDT, Lecture Notes in Computer Science, Vol. 1186, Springer, Berlin, 1997, pp. 56–70.
[16] S.S. Cosmadakis, P. Kanellakis, Parallel evaluation of recursive queries, in: PODS, 1986, pp. 280–293.
[17] O.M. Duschka, Query planning and optimization in information integration, Ph.D. Thesis, Computer Science Department, Stanford University, 1997.
[18] O.M. Duschka, M.R. Genesereth, Answering recursive queries using views, in: PODS, 1997, pp. 109–116.
[19] D. Florescu, A. Levy, D. Suciu, K. Yagoub, Optimization of run-time management of data intensive web-sites, in: Proc. of VLDB, 1999, pp. 627–638.
[20] A. Gupta, Y. Sagiv, J.D. Ullman, J. Widom, Constraint checking with partial information, in: PODS, 1994, pp. 45–55.
[21] G. Grahne, A.O. Mendelzon, Tableau techniques for querying information sources through global schemas, in: ICDT, 1999, pp. 332–347.
[22] L.M. Haas, D. Kossmann, E.L. Wimmers, J. Yang, Optimizing queries across diverse data sources, in: Proc. of VLDB, 1997, pp. 276–285.
[23] Z. Ives, D. Florescu, M. Friedman, A. Levy, D. Weld, An adaptive query execution engine for data integration, in: SIGMOD, 1999, pp. 299–310.
[24] A. Klug, On conjunctive queries containing inequalities, J. ACM 35 (1) (1988) 146–160.
[25] P.G. Kolaitis, D.L. Martin, M.N. Thakur, On the complexity of the containment problem for conjunctive queries with built-in predicates, in: PODS, 1998, pp. 197–204.
[26] A. Levy, Answering queries using views: a survey, Technical Report, Computer Science Department, Washington University, 2000.
[27] A. Levy, A.O. Mendelzon, Y. Sagiv, D. Srivastava, Answering queries using views, in: PODS, 1995, pp. 95–104.
[28] A. Levy, A. Rajaraman, J.J. Ordille, Querying heterogeneous information sources using source descriptions, in: Proc. of VLDB, 1996, pp. 251–262.

[29] A. Levy, Y. Sagiv, Queries independent of updates, in: Proc. of VLDB, 1993, p. 171181.
[30] P. Mitra, An algorithm for answering queries efficiently using views, in: Proc. of the Australasian Database Conf., 2001.
[31] R. Pottinger, A. Levy, A scalable algorithm for answering queries using views, in: Proc. of VLDB, 2000.
[32] X. Qian, Query folding, in: 12th Internat. Conf. on Data Engineering, 1996.
[33] Y. Sagiv, Optimizing datalog programs, Foundations of Deductive Databases and Logic Programming, 1988, pp. 659–698.
[34] Y. Saraiya, Subtree elimination algorithms in deductive databases, Ph.D. Thesis, Computer Science Department, Stanford University, 1991.
[35] D. Theodoratos, T. Sellis, Data warehouse configuration, in: Proc. of VLDB, 1997.
[36] J.D. Ullman, Principles of Database and Knowledge-base Systems, Vol. II: The New Technologies, Computer Science Press, New York, 1989.
[37] J.D. Ullman, Information integration using logical views, in: ICDT, 1997, pp. 19–40.
[38] R. van der Meyden, The complexity of querying indefinite data about linearly ordered domains, in: PODS, 1992.
[39] R. van der Meyden, The complexity of querying infinite data about linearly ordered domains, J. Comput. System Sci. 54 (1) (1997) 113–135.
[40] X. Zhang, Z.M. Ozsoyoglu, Some results on the containment and minimization of (in) equality queries, Inform. Process. Lett. (1994).