Efficient Complete Local Tests for Conjunctive Query Constraints with Negation

Nam Huyn *

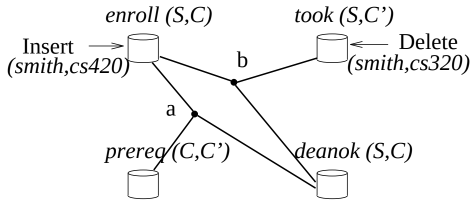
Stanford University

Abstract. We consider the problem of incrementally checking global integrity constraints without using all the relations under constraint. In many application areas such as collaborative design, mobile computing and enterprise information systems, total data availability cannot be assumed. Even if all base data is available, some of it may incur such a high cost that its use should only be considered as a last resort. Without looking at all the base data, how can one meaningfully check a constraint for violation? When the constraint is known to be satisfied prior to the update, the state of the relations that are available (aka local) can in principle be used to infer something about the relations that are not available (aka remote). This observation is the basis for the existence of tests that guarantee that data integrity is preserved under a given update, without looking at all the base data.
In order to make integrity maintenance *practical*, the challenge is to find those tests that are *most general* (we call them *Complete Local Tests* or CLT's in short) and that are *efficient* to generate and execute. This paper addresses the problem of finding efficient CLT's for an important class of constraints that are very common in practice: constraints expressible as conjunctive queries with negated subgoals (abbreviated $\mathrm{CQC}^{\neg}$.) We show that for single updates, all $\mathrm{CQC}^{\neg}$ constraints admit a CLT that can be expressed in nonrecursive $\mathrm{Datalog}^{\neg}$ when the predicates for the remote relations are not repeated in the constraint query. We then extend this result to a larger class of constraints and to certain sets of updates.

1 Introduction

The problem of incrementally checking global integrity constraints without using all the participating relations is important in many global information systems where base data is not totally available. In some cases, information access may be restricted for security reasons. In other cases, remote data sources may be disconnected temporarily due to mobility requirements. Cost may also be a factor in deciding not to use certain information sources if not absolutely needed. In some cases, we can optimize constraint maintenance in a distributed environment by first performing local checks for potential constraint violations before considering more expensive global checks. Consider a university information system as depicted in Fig. 1. While the databases are independently managed at their own site, they are subject to a global integrity constraint that students may

* This work was supported by ARO grant DAAH04–95–1–0192.



enroll (S,C)	student S was officially enrolled in course C
prereq (C,C')	course C has course C' as a prerequisite
took (S,C')	student S has taken course C'
deanok (S,C)	student S has the dean's approval to enroll in C

Fig. 1. University information system.

not enroll in a course unless they have either taken all the prerequisites for the course or obtained a special approval from the dean to enroll. This enrollment policy can be stated by the following integrity constraint:

$$(\forall\ S, C, C')\ enroll(S, C) \wedge prereq(C, C') \Rightarrow took(S, C') \vee deanok(S, C) \quad (1)$$

The databases may be disconnected due to network failure during which updates are allowed as long as the system remains globally consistent. Consider the insertion of a new tuple $enroll(smith, cs420)$. Let us see how we can guarantee that the insertion is safe in each of the following three scenarios.

1. Node a fails and relation $prereq$ becomes inaccessible. It is safe to enroll $smith$ in $cs420$ if he already has approval from the dean. Alternatively, consider the students who were enrolled in $cs420$ without using dean's approval (assuming there is one). Anyone of these students must have satisfied all the prerequisites for $cs420$. All these prerequisites must be among $\mathcal{P}$, the courses that all these students have taken. $\mathcal{P}$ is shown in Fig. 2 with hashed lines. If $smith$ has taken all courses in $\mathcal{P}$, he must have satisfied all the prerequisites for $cs420$.

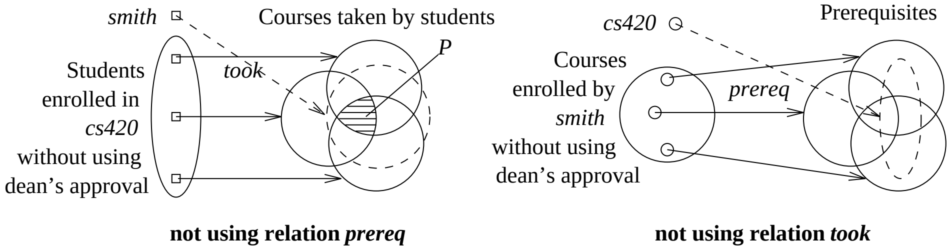


Fig. 2. Preserving data integrity under different scenarios.

2. Node b fails and relation $took$ becomes inaccessible. Again, approval from the dean is one way to safely enroll $smith$. Alternatively, we want to make sure $smith$ has taken all the prerequisite courses for $cs420$. Obviously, we cannot directly check this condition since relation $took$ is not available. Fortunately, there is an indirect method that considers the prerequisites for all courses

smith was enrolled in without using the dean's approval. If these prerequisites include all of $cs420$'s prerequisites, we can conclude that *smith* has indeed taken all $cs420$'s prerequisites. Fig. 2 illustrates this test.

3. Both nodes fail and none of the relations *prereq*, *took* and *deanok* is accessible. There is no way to ensure *smith*'s enrollment is legitimate.

While these tests obviously guarantee that data integrity is preserved under a given update, we also wish them to be as general as possible. Indeed, a degenerate test procedure that always results in predicting potential violation is certainly sound but too conservative to be of practical use. The challenge is to find the *most general* of these tests when they exist. Such most general tests are called *Complete Local Tests* (or just Complete Tests or CLT's in short) and can be formalized as follows:

- A CLT only looks at the instance of the local [2] base relations and the update instance.
- When the test is satisfied, global consistency is guaranteed to be preserved, regardless of the instance of remote relations. When it is not satisfied, there is some instance of the remote relations such that global consistency is not preserved.

Constraints of the form $(\forall \bar{U})(g_1 \wedge \ldots \wedge g_m \Rightarrow h_1 \vee \ldots \vee h_n)$, such as (1), are very common in practice. These constraints are used to specify that for a combination of data $(\bigwedge_i g_i)$ to be legitimate, it must have a "cover" among the h_j's (i.e., a tuple in the relation for some h_j that legitimizes the combination). These constraints have an equivalent query form in which the database is queried for violations, i.e., combinations of data that have no cover. The database is consistent if the constraint query returns no answers. In query form, the constraints of interest are represented as Datalog queries $panic$:- $g_1, \ldots, g_m, \neg h_1, \ldots, \neg h_n$. For this reason, we call them *Conjunctive-Query Constraints with Negation*, or $\mathrm{CQC}^{\neg}$ in short. Besides the query form for $\mathrm{CQC}^{\neg}$'s, the logic form will also be used to explain our reasoning.

This paper mainly considers the problem of finding not only most general tests for constraints whose query involves negation in general and for $\mathrm{CQC}^{\neg}$'s in particular, but also tests that can be *efficiently* implemented such as first order queries. The $\mathrm{CQC}^{\neg}$ class is shown in Fig. 3 as the shaded oval. Because of the presence of negation in the query, many questions that have practical significance are not immediately obvious. Do $\mathrm{CQC}^{\neg}$'s admit test procedures that are complete and that can be efficiently generated? Do $\mathrm{CQC}^{\neg}$'s admit CLT's that can be efficiently executed, i.e., in time polynomial in the size of the local relations? Can these CLT's be expressed as first order queries so they can be executed using conventional query-evaluation engines? In this paper, we establish the following results:

[2] For the rest of this paper, we choose to use the local/remote terminology purely to keep it consistent with [G94]. We say that a relation is "local" merely to indicate that it is available for use but not necessarily that is is physically local. Similar, "remote" is synonymous to "not available" for use.

- For single updates to local relations, all CQC$^\neg$'s admit complete local tests that can be expressed as nonrecursive Datalog queries with negation, provided each remote predicate has a single occurrence in the constraint query (shown as a dashed arc labeled "Our work" in Fig. 3). Using different techniques we recently developed but have not described in this paper, our results are extended to cover the entire conjunctive-query (CQ) class by lifting the restriction of no repeated predicates.
- These results are extended to a larger class of constraints and arbitrary sets of insertion updates and to certain sets of deletion updates to the local relations.
- All solution tests can be generated in time at most exponential in the size of the constraints. We give the solutions in a form that makes their expression in (safe) nonrecursive Datalog$^\neg$ (and other traditional query languages such as SQL) straightforward.

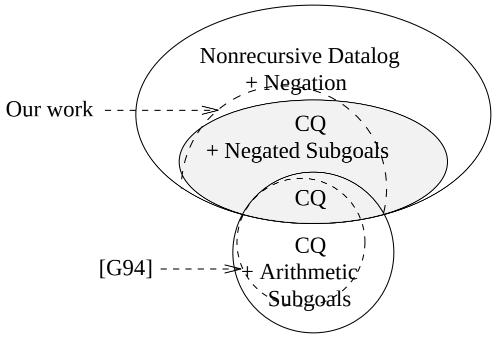


Fig. 3. Constraint classes for the CLT problem

Related Work. The notions of local tests and completeness were first introduced in [G94, GSUW94] and were embodied in an actual system supporting collaborative design in building construction [TH93]. The most powerful result for CLT's that can be found in [G94] applies to a subset of conjunctive-query constraints with arithmetic and handles only insertion updates and single local subgoals (shown as a dashed circle in Fig. 3). Negation was considered in [G94] but only for local tests that are not complete. The approach used in [G94] for finding complete tests is based on containment of conjunctive queries with arithmetic comparisons, and only in some restricted cases, a closed form solution for the CLT's (as queries) can be obtained by expanding and simplifying the containment tests.

In [LS93] a larger class of Datalog programs with negation was considered, but for a different though closely related problem: the problem of detecting queries independent of updates not looking at any relations. The problem of consistency maintenance considered in this paper is also closely related to the problem of view self-maintenance (see [TB88, GB95, H96b]) as follows: checking CQC$^\neg$ constraints reduces to checking if a given view instance is independent of a given update, but the latter problem is still open for CQ views with negation and with an arbitrary number of remote predicates. In fact, a subclass of this problem, namely that of checking if a given view instance gains new tuples under a given update to some local relation, reduces to our problem and can be solved using the results presented in this paper.

Alternative to containment-based approach. The problem of maintaining CQC$^{\neg}$'s can be reduced to that of testing containment of unions of conjunctive queries with negation and $\neq$ comparison. For instance, when l and g are available (with L and G as extensions) but h and r are not, maintaining the constraint $panic$:- $l(X), \neg g(X,Z), \neg h(X,Z), r(Z)$. under the insertion of $l(a)$ reduces to testing $Q_a \subseteq \bigcup_{t \in L} Q_t$, where Q_s denotes the boolean query $panic$:- $\neg g(s,Z), \neg h(s,Z), r(Z)$. To eliminate predicate g (whose extension is *given*) from Q_s, the trick is to rewrite Q_s as

$$panic \text{ :- } \neg h(s,Z), r(Z), \bigwedge_{(x,z) \in G} (s \neq x \vee Z \neq z).$$

which in turn can be expanded into a union of conjunctive queries with negation and $\neq$. [LS93] gave an algorithm for testing such containments, which unfortunately has a complexity exponential in the size of the given relations in our problem (the queries have an exponential number of rules, and the rules have a number of subgoals the size of the given relations). Furthermore, it is not clear if one can always turn the algorithm for testing containment into a first order query independent of the extension of the given relations. While a containment-based approach cannot be easily applied to efficiently maintain CQC$^{\neg}$'s, the "university" example used earlier seems to suggest that an alternative but more direct approach can be used instead to find CLT's. The approach we offer in this paper essentially amounts to compile-time generation of runtime tests.

Paper Outline. The rest of this paper is organized as follows. In the next section, we provide some basic concepts on maintaining CQC$^{\neg}$ constraints, state assumptions and present our notation and terminology conventions. In Section 3, we illustrate our approach to the problem of finding complete tests by showing the concepts and the reasoning used in deriving complete local tests for our running example. We then present the main results of this paper in Sections 4 and 5. Finally in Section 6, we summarize our work and discuss possible extensions.

2 Preliminaries

Datalog$^{\neg}$. In this paper, integrity constraints (and their complete tests if any) are modeled as queries expressed by Datalog$^{\neg}$ programs. A Datalog$^{\neg}$ program is a collection of Horn rules that may have negated subgoals in their bodies. These rules are assumed to be *safe* (see [Ull88]), and when a program has both recursion and negation, stratified negation is assumed. Predicates used in a Datalog$^{\neg}$ program are partitioned into EDB predicates and IDB predicates. EDB predicates are those that only occur in rule bodies and represent base relations that are under constraint. Among the IDB predicates, there is a distinguished predicate called the *query predicate* that generates answers for the query. The following variable naming convention will be used in all Datalog$^\neg$ programs as well as in logic sentences: variable names always start in upper case, constant names in lower case.

Constraints. We mainly focus on constraints expressed by conjunctive queries with negation (CQC$^\neg$), that is, Datalog$^\neg$ programs consisting of a single rule of the form $panic$:- $g_1, \ldots, g_m, \neg h_1, \ldots, \neg h_n.$, where the g_i's and h_i's denote positive literals with EDB predicates. In order to ensure safe use of negation, variables used in the h_i's must occur among the g_i's. The body in the rule above is called the *constraint query*. When the constraint query has an answer, the constraint is violated.

Since some of the EDB relations are local and others remote, we introduce a notation that makes the local vs remote distinction explicit. This notation also makes variable use in subgoals explicit. Thus, for the remainder of this paper, CQC$^\neg$'s are represented as

$$panic \text{ :- } P(\bar{X}, \bar{Y}), \bigwedge_{i \in L} \neg Q_i(\bar{X}_i, \bar{Y}_i, \bar{Z}_i), \bigwedge_{j \in M} \neg \boldsymbol{Q}_j(\bar{X}_j, \bar{Y}_j, \bar{Z}_j), \boldsymbol{R}(\bar{Y}, \bar{Z}). \qquad (2)$$

- $P(\bar{X}, \bar{Y})$ denotes a conjunction of zero or more positive local subgoals (i.e. subgoals with local predicates), and $R(\bar{Y}, \bar{Z})$ a conjunction of zero or more positive remote subgoals. $\bar{X}$ denotes the set of variables that are used in positive local subgoals but not in positive remote subgoals, $\bar{Y}$ denotes the set of variables used in some positive local subgoal and some positive remote subgoal, and $\bar{Z}$ denotes the set of variables that are used in positive remote subgoals but not in positive local subgoals. Thus $\bar{X}$,$\bar{Y}$,$\bar{Z}$ are mutually disjoint. Note the use of boldface in the constraint query (2) to emphasize the fact that the subgoals in R are remote.
- The negated subgoals are partitioned into two subsets indexed by L and M: L indexes the negated local subgoals, that is, for $i \in L$, $Q_i(\bar{X}_i, \bar{Y}_i, \bar{Z}_i)$ denotes a subgoal with some local predicate; M indexes the negated remote subgoals, that is, for $j \in M$, $Q_j(\bar{X}_j, \bar{Y}_j, \bar{Z}_j)$ denotes a subgoal with some remote predicate. Also note the use of boldface in (2) for these remote Q_j's.
- $\bar{X}_i \subseteq \bar{X}$, $\bar{Y}_i \subseteq \bar{Y}$, $\bar{Z}_i \subseteq \bar{Z}$ are assumed for all $i \in L$ and $i \in M$ to ensure safe use of negation.

Unless explicitly stated otherwise, we use $\bar{a}$ (resp. $\bar{b}$, $\bar{c}$) to denote a vector of constants of the same arity as $\bar{X}$ (resp. $\bar{Y}$,$\bar{Z}$). When $\bar{X}_i \subseteq \bar{X}$, $\bar{a}_i$ denotes the projection of $\bar{a}$ onto the variables in $\bar{X}_i$. We will use $\bar{X} =_M \bar{X}'$ as a shorthand for $\bigwedge_{j \in M} \bar{X}_j = \bar{X}'_j$, representing the fact that the two vectors $\bar{X}$ and $\bar{X}'$ agree over the variables in all $\bar{X}_j$'s. Given some index $h \in L$, we will use L_h^+ to denote the set of indices $i \in L$ such that $\bar{Z}_i \subseteq \bar{Z}_h$. L_h^+ represents the negated local subgoals that do not use Z-variables other than those in $\bar{Z}_h$. $L_h^- (= L - L_h^+)$ denotes the indices for the remaining negated local subgoals.

Updates. Given a constraint and accessibility information about the EDB predicates as in (2), we only consider insertion and deletion updates to the local EDB relations. These updates are modeled as a set ΔP (resp. ΔQ_i) of tuples to be added to or deleted from the subquery $P(\bar{X}, \bar{Y})$ (resp. local subquery $Q_i(\bar{X}_i, \bar{Y}_i, \bar{Z}_i)$). For no loss of generality, we assume that when inserting a given ΔP into P, no tuples from ΔP is already in P. Similarly, when deleting a given ΔP from P, we assume all tuples from ΔP are already in P. The same assumption applies to the local ΔQ_i.

We will say that an update is *safe* (with respect to given local relations) if the update preserves global data consistency (with respect to a given constraint).

3 Extended example

In this section, we will go through the process of finding a complete local test for our running example. We will present the formal results for the general case in the next section. Recall from Section 1 the problem of finding complete tests for constraint (1) repeated here for convenience

$$(\forall\ S, C, C')\ enroll(S, C) \wedge prereq(C, C') \Rightarrow took(S, C') \vee deanok(S, C) \qquad (3)$$

when relation *prereq* is remote. We first consider a simple insertion and later a simple deletion.

Characterizing when a simple insertion is safe. Suppose we want to insert $enroll(smith, cs420)$. To safely enroll *smith* in $cs420$, we must ensure that he has either obtained approval from the dean or taken all the prerequisites. In other words, a *cover* must be found for any tuple ($smith$,$cs420$,C') where C' is a prerequisite for $cs420$. This condition is formalized as follows (by substituting S for *smith* and C for $cs420$ in (3)):

$$[(\forall C')\ prereq(cs420, C') \Rightarrow took(smith, C')] \vee deanok(smith, cs420) \qquad (4)$$

The problem is that relation *prereq* is not available and we cannot directly look up the prerequisites for $cs420$. In order to eliminate the "unknown" $prereq(cs420, C')$ from (4), we must be able to "bound" $cs420$'s prerequisites or else, our "adversary" will be able to prevent *smith* from legally enrolling in $cs420$ by inventing prerequisites *smith* never took. We have just used a simple *adversary argument* to deal with the unknown, in which an imaginary adversary is trying to cause constraint violations by controlling the unknown. The adversary is able to do so if the following *boundability condition* is not satisfied (i.e., when every student uses dean's approval to enroll in $cs420$, the prerequisites no longer matter):

$$(\exists S)\ [enroll(S, cs420) \wedge \neg deanok(S, cs420)] \qquad (5)$$

A key observation is that the adversary *is not totally free* to choose a state for *prereq* and must still respect the constraint that all past enrollments in $cs420$

were legal (by substituting C for $cs420$ in (3)):

$$(\forall S)\ \ enroll(S, cs420) \wedge \neg deanok(S, cs420) \\ \Rightarrow (\forall C')\ [prereq(cs420, C') \Rightarrow took(S, C')] \tag{6}$$

Boundability condition (5) amounts to requiring that constraint (6) is not vacuously satisfied. Constraint (6) also gives the bound we call $possible\text{-}cs420\text{-}prereq(C')$ representing all classes commonly taken by students without using dean's approval and formalized as:

$$possible\text{-}cs420\text{-}prereq(C') \stackrel{\text{def}}{\equiv} \\ (\forall S)\ enroll(S, cs420) \wedge \neg deanok(S, cs420) \Rightarrow took(S, C') \tag{7}$$

This bound is tight and the complete test for safe insertion is obtained by replacing unknown $prereq(cs420, C')$ in (4) with the bound $possible\text{-}cs420\text{-}prereq(C')$, and by attaching the boundability condition:

$$deanok(smith, cs420) \vee \\ (5) \wedge (\forall C')\ possible\text{-}cs420\text{-}prereq(C') \Rightarrow took(smith, C') \tag{8}$$

Ensuring test is finitely evaluable. Condition (8) would be easy to implement as a Datalog$^\neg$ program (or in any traditional query language) if we know how to *finitely* evaluate $possible\text{-}cs420\text{-}prereq(C')$. The problem is that the current definition of $possible\text{-}cs420\text{-}prereq(C')$ in (7) is not safe. If every student uses dean's approval to enroll in $cs420$, then relation $possible\text{-}cs420\text{-}prereq$ would not be finite. Without changing its logical meaning, $possible\text{-}cs420\text{-}prereq(C')$ can simply be redefined to be safe by incorporating the boundability condition and by making it explicit that the values for C' actually come from relation $took$:

$$possible\text{-}cs420\text{-}prereq(C') \stackrel{\text{def}}{\equiv} \\ (\exists S)\ enroll(S, cs420) \wedge \neg deanok(S, cs420) \wedge took(S, C') \wedge \\ (\forall S)\ enroll(S, cs420) \wedge \neg deanok(S, cs420) \Rightarrow took(S, C')$$

The complete test (8) for safely inserting $enroll(smith, cs420)$ is expressed by the following nonrecursive Datalog$^\neg$ program using `safeinsert` as the query predicate:

```
  enroll-cs420-no-dean(S) :- enroll(S,cs420), ¬deanok(S,cs420).
                union(C') :- enroll-cs420-no-dean(S), took(S,C').
  not-a-cs420-prereq(C') :- union(C'), enroll-cs420-no-dean(S),
                            ¬took(S,C').
possible-cs420-prereq(C') :- union(C'), ¬not-a-cs420-prereq(C').
              notcovered :- possible-cs420-prereq(C'),
                            ¬took(smith,C').
              safeinsert :- deanok(smith,cs420).
              safeinsert :- enroll-cs420-no-dean(S),
                            ¬notcovered.
```

Deriving complete test for a simple deletion. Now suppose we want to delete *took*(*smith*, *cs*320). For a deletion to be safe, we want to avoid situations where the tuple to be deleted is the sole way to legitimize the existence of certain tuples in other relations. If *smith* was enrolled in a course that has *cs*320 as prerequisite, we must ensure that he was able to do so only with the dean's approval. In other words, any combination (*smith*, C, *cs*320) that uses *took*(*smith*, *cs*320) as a cover, must have an *alternative cover* to be found in *deanok*, the only choice of local relations left. This condition is formalized as

$$(\forall C)\ [\mathit{enroll}(\mathit{smith}, C) \wedge \mathit{prereq}(C, \mathit{cs}320)] \Rightarrow \mathit{deanok}(\mathit{smith}, C) \tag{9}$$

Since *prereq*(C, *cs*320) is unknown, the same idea of bounding courses having *cs*320 as prerequisite applies here as well, using the fact that prior to the deletion, there was no enrollment violations involving courses having *cs*320 as prerequisite:

$$(\forall C)\ \mathit{prereq}(C, \mathit{cs}320) \Rightarrow [(\forall S)\ \mathit{enroll}(S, C) \Rightarrow \mathit{took}(S, \mathit{cs}320) \vee \mathit{deanok}(S, C)]$$

The bound we call *possible-cs*320*-prereqfor*(C), is found on the right hand side of the implication above:

$$\begin{aligned}&\mathit{possible\text{-}cs320\text{-}prereqfor}(C) \stackrel{\text{def}}{\equiv}\\&(\forall S)\ \mathit{enroll}(S, C) \Rightarrow \mathit{took}(S, \mathit{cs}320) \vee \mathit{deanok}(S, C)\end{aligned} \tag{10}$$

By eliminating unknown *prereq*(C, *cs*320) from (9) using its bound (10), we obtain the following complete test for the safe deletion of *took*(*smith*, *cs*320):

$$(\forall C)\ \mathit{enroll}(\mathit{smith}, C) \wedge \mathit{possible\text{-}cs320\text{-}prereqfor}(C) \Rightarrow \mathit{deanok}(\mathit{smith}, C)$$

This test can be directly implemented as the following nonrecursive Datalog$^\neg$ program with `safedelete` as the query predicate:

```
          enroll-no-dean(S,C) :- enroll(S,C), ¬deanok(S,C).
                    forbid(C) :- enroll-no-dean(S,C),
                                 ¬took(S,cs320).
possible-cs320-prereqfor(C) :- enroll-no-dean(S,C), ¬forbid(C).
                       unsafe :- enroll-no-dean(smith,C),
                                 possible-cs320-prereqfor(C).
                   safedelete :- ¬unsafe.
```

4 General Case of Single Updates

We now present solutions for the problem of finding complete tests for general CQC$^\neg$ constraints, that is, constraints of the form

$$\mathit{panic} :- P(\bar{X}, \bar{Y}), \bigwedge_{i \in L} \neg Q_i(\bar{X}_i, \bar{Y}_i, \bar{Z}_i), \bigwedge_{j \in M} \neg \boldsymbol{Q}_j(\bar{X}_j, \bar{Y}_j, \bar{Z}_j), \boldsymbol{R}(\bar{Y}, \bar{Z}).$$

under the restriction that no remote predicate has multiple occurrences among the subqueries R and Q_j for $j \in M$, and under the restriction of single updates, i.e., when only one of the subqueries P or Q_i's for $i \in L$ changes, and only one tuple is either added to or removed from the subquery under change. We will successively consider the following three cases that have nontrivial solutions: (1) P vacuous (i.e. P has no conjuncts); (2) R vacuous; and (3) P and R nonvacuous (i.e. each has one or more conjuncts). The only remaining case that we are not considering here (for single updates) has a trivial solution which we can immediately state now: when there are no negated subgoals, inserting $P(\bar{a}, \bar{b})$ is safe if and only if $(\exists \bar{X})P(\bar{X}, \bar{b})$ (see [H96atr]) and the net effect of removing anything from P is *always safe*. Due to space limitation, we omit all formal proofs and implementations in Datalog$^{\neg}$, which can be found in [H96atr]. We prove sufficiency of all the test conditions in this section by case analysis, and their necessity by case analysis and counterexample construction.

4.1 No positive local subgoals

When there are no positive local subgoals, the constraint takes on a special form where there are neither X-variables nor Y-variables:

$$panic \text{ :- } \bigwedge_{i \in L} \neg Q_i(\bar{Z}_i), \bigwedge_{j \in M} \neg \boldsymbol{Q}_j(\bar{Z}_j), \boldsymbol{R}(\bar{Z}).$$

Single insertions into the local Q_i's are always safe since the constraint query is antimonotonic in these Q_i's (i.e., the constraint query never gain tuples when tuples are added to the Q_i's). The following theorem gives a CLT solution for the general case of no positive local subgoals under single deletions.

Theorem 1. *(Deletion) Let $h \in L$ and let $\bar{c}$ be a vector of constants of the same arity as $\bar{Z}_h$. The following is a CLT for the safe deletion of $Q_h(\bar{c})$:*

$$\bigvee_{i \in L_h^+, i \neq h} Q_i(\bar{c}_i).$$

Intuitively, we want to make sure that any tuple $(\bar{Z})$ that uses $Q_h(\bar{c})$ as a cover, has an alternative cover to be found among the local Q_i's other than Q_h. We cannot count on the remote Q_j's to provide cover since they are controlled by the adversary. Also, since the variables outside of $\bar{Z}_h$ are totally controlled by the adversary, we are reduced to find cover among the local Q_i's that only use variables in $\bar{Z}_h$.

4.2 No positive remote subgoals

When there are no positive remote subgoals, the constraint takes on a special form where there are neither Y-variables nor Z-variables:

$$panic \text{ :- } P(\bar{X}), \bigwedge_{i \in L} \neg Q_i(\bar{X}_i), \bigwedge_{j \in M} \neg \boldsymbol{Q}_j(\bar{X}_j).$$

The following theorems give solution CLT for the general case of no positive remote subgoals under a single insertion and a single deletion.

Theorem 2. *(Insertion) The following is a CLT for the safe insertion of $P(\bar{a})$:*

$$[\bigvee_{i \in L} Q_i(\bar{a}_i)] \vee (\exists \bar{X})[P(\bar{X}) \wedge (\bar{X} =_M \bar{a}) \wedge \bigwedge_{i \in L} \neg Q_i(\bar{X}_i)]$$

When a local cover cannot be found for the newly inserted $P(\bar{a})$, a remote cover must be found, albeit not directly. The trick here is to look for tuples in P that have no local cover (and thus must have used a remote cover), and that are "indistinguishable" from tuple $(\bar{a})$ from the remote point of view.

Theorem 3. *(Deletion) Let $h \in L$ and let $\bar{a}$ be a vector of constants of the same arity as $\bar{X}_h$. The following is a CLT for the safe deletion of $Q_h(\bar{a})$:*

$$\begin{aligned} (\forall \bar{X}) \quad & P(\bar{X}) \wedge \bar{X}_h = \bar{a} \wedge \bigwedge_{i \in L, i \neq h} \neg Q_i(\bar{X}_i) \\ & \Rightarrow (\exists \bar{X}')(P(\bar{X}') \wedge (\bar{X}' =_M \bar{X}) \wedge \bigwedge_{i \in L} \neg Q_i(\bar{X}'_i)) \end{aligned}$$

Any tuple $(\bar{X})$ in P that uses $Q_h(\bar{a})$ as a cover prior to its deletion and that has no other local covers, must have a remote cover which can be indirectly found if we can find some tuple $(\bar{X}')$ in P that has no local cover and that is indistinguishable from $(\bar{X})$ from the remote point of view.

4.3 Both positive local and remote subgoals present

Generalizing on the extended example in Section 3, the following theorems give the complete test solutions for the general case when both positive local and remote subgoals are present. The reader is warned that the test expressions are rather complex. This complexity is partly due to the generality of the results, which unify simpler results from more specialized subcases. Immediately following the theorems, we will briefly give an intuitive account of various subexpressions in the tests and point out opportunities when the test expressions can be drastically simplified.

In the following theorem for insertion, we use $\bar{Z}_I$ to denote the union of all variables from $\bar{Z}_i$ for all $i \in I$, for a given set I of indices.

Theorem 4. *(Insertion) The following is a CLT for the safe insertion of $P(\bar{a},\bar{b})$:*

$$\begin{aligned} & \overbrace{\bigvee_{i \in L, \bar{Z}_i = \emptyset} Q_i(\bar{a}_i, \bar{b}_i)}^{(i)} \vee [\ \overbrace{(\exists \bar{X})\ (P(\bar{X}, \bar{b}) \wedge \bar{X} =_M \bar{a})}^{(ii)} \wedge \\ & \qquad \bigwedge_{I \subseteq L, I \neq \emptyset} \underbrace{(\forall \bar{Z}_I)\ [\phi^I_{\bar{a}\bar{b}}(\bar{Z}_I) \Rightarrow \bigvee_{i \in L, \bar{Z}_i \subseteq \bar{Z}_I} Q_i(\bar{a}_i, \bar{b}_i, \bar{Z}_i)]]}_{(iii)} \qquad (11) \end{aligned}$$

where $\phi^I_{\bar{a}\bar{b}}(\bar{Z}_I)$ *is defined as:*

$$\begin{aligned}&\bigwedge_{i\in I}(\exists \bar{X})\ [P(\bar{X},\bar{b}) \wedge (\bar{X} =_M \bar{a}) \wedge Q_i(\bar{X}_i,\bar{b}_i,\bar{Z}_i)]\\ &\wedge(\forall \bar{X})\ [P(\bar{X},\bar{b}) \wedge (\bar{X} =_M \bar{a}) \Rightarrow \bigvee_{i\in I} Q_i(\bar{X}_i,\bar{b}_i,\bar{Z}_i)]\end{aligned}$$

Intuitively, we want to find a cover for any tuple $(\bar{a},\bar{b},\bar{Z})$ such that $R(\bar{b},\bar{Z})$. The local covers provided by subexpression *(i)* in (11) are the simplest since no Z-variables are involved. When these simple local covers fail, the situation becomes more complicated since we now have to not only find cover for these Z-values but also bound them. Roughly speaking, subexpression *(ii)* represents the boundability condition (analogous to (5) in the extended example), and the $\phi^I_{\bar{a}\bar{b}}(\bar{Z}_I)$'s are the bounds (analogous to (7) in the extended example). The implication *(iii)* essentially provides cover for the Z-values once bounded.

Note how all nonempty subsets $I \subseteq L$ are involved in testing for inclusion of the Z-values. This extra complexity is needed to make sure that the $\phi^I_{\bar{a}\bar{b}}(\bar{Z}_I)$ queries can be finitely evaluated, especially when the negated subgoals Q_i's are allowed to use arbitrarily different subsets of Z-variables. To appreciate this small but important point, consider the formula $(\forall\ Z_1,Z_2)[(\forall X)(p(X) \Rightarrow (q_1(X,Z_1) \vee q_2(X,Z_2)))] \Rightarrow [r_1(Z_1) \vee r_2(Z_2)]$. Even if relation p is nonempty, this formula is not safe since the number of pairs (Z_1,Z_2) that satisfy the left hand side of the top level implication may be unbounded. Since we only need to bound their projections onto Z_1 or Z_2, a solution is to generate the values *for each subset* of the Z-variables using the following safe queries

$$\begin{aligned}\varphi_1(Z_1) &\stackrel{\text{def}}{\equiv} (\exists X)(p(X) \wedge q_1(X,Z_1)) \wedge (\forall X)(p(X) \Rightarrow q_1(X,Z_1))\\ \varphi_2(Z_2) &\stackrel{\text{def}}{\equiv} (\exists X)(p(X) \wedge q_2(X,Z_2)) \wedge (\forall X)(p(X) \Rightarrow q_2(X,Z_2))\\ \varphi_{12}(Z_1,Z_2) &\stackrel{\text{def}}{\equiv} (\exists X)(p(X) \wedge q_1(X,Z_1)) \wedge (\exists X)(p(X) \wedge q_2(X,Z_2)) \wedge\\ &\qquad (\forall X)(p(X) \Rightarrow (q_1(X,Z_1) \vee q_2(X,Z_2)))\end{aligned}$$

and test each of these subsets using the following equivalent but safe formula:

$$\begin{aligned}&(\forall\ Z_1)\ [\varphi_1(Z_1) \Rightarrow r_1(Z_1)] \wedge\\ &(\forall\ Z_2)\ [\varphi_2(Z_2) \Rightarrow r_2(Z_2)] \wedge\\ &(\forall\ Z_1,Z_2)\ [\varphi_{12}(Z_1,Z_2) \Rightarrow (r_1(Z_1) \vee r_2(Z_2))]\end{aligned}$$

The following theorem for deletion uses the notation $\bar{Z}_I =_{I,h} \bar{c}$ (for some $I \subseteq L^-_h$, some $h \in L$, and some vector $\bar{c}$ of constants of same arity as $\bar{Z}_h$) that needs to be clarified: the left hand side represents constant values for variables in $\bar{Z}_I$; the right hand side represents constant values for variables in $\bar{Z}_h$; when $\bar{Z}_I$ and $\bar{Z}_h$ have no variables in common, the comparison is vacuously satisfied; otherwise, there must be agreement over the common variables.

Theorem 5. *(Deletion) Let $h \in L$ and let $\bar{a}$ (resp. $\bar{b}$, $\bar{c}$) be a vector of constants of the same arity as $\bar{X}_h$ (resp. $\bar{Y}_h$, $\bar{Z}_h$). The following is a CLT for the safe deletion of $Q_h(\bar{a}, \bar{b}, \bar{c})$:*

$$\begin{aligned}
& (\forall\, \bar{X}, \bar{Y})[\ [P(\bar{X}, \bar{Y}) \wedge \eta_{\bar{a}\bar{b}\bar{c}}(\bar{X}, \bar{Y})] \\
& \qquad \Rightarrow (\exists \bar{X}')(\psi_{\bar{b}\bar{c}}(\bar{X}', \bar{Y}) \wedge \bar{X}' =_M \bar{X})] \wedge \\
\bigwedge_{I \subseteq L_h^-, I \neq \emptyset} & (\forall\, \bar{X}, \bar{Y}, \bar{Z}_I)[\ [P(\bar{X}, \bar{Y}) \wedge \eta_{\bar{a}\bar{b}\bar{c}}(\bar{X}, \bar{Y}) \wedge \xi^I_{\bar{b}\bar{c}}(\bar{X}, \bar{Y}, \bar{Z}_I)] \\
& \qquad \Rightarrow \bigvee_{i \in L_h^-, \bar{Z}_i \subseteq \bar{Z}_I} Q_i(\bar{X}_i, \bar{Y}_i, \bar{Z}_i)]
\end{aligned} \tag{12}$$

where $\eta_{\bar{a}\bar{b}\bar{c}}$, $\psi_{\bar{b}\bar{c}}$ and $\xi^I_{\bar{b}\bar{c}}$ are defined by:

$$\begin{aligned}
\eta_{\bar{a}\bar{b}\bar{c}}(\bar{X}, \bar{Y}) &\stackrel{\text{def}}{\equiv} \bar{X}_h = \bar{a} \wedge \bar{Y}_h = \bar{b} \wedge \textstyle\bigwedge_{i \in L_h^+, i \neq h} \neg Q_i(\bar{X}_i, \bar{Y}_i, \bar{c}_i) \\
\psi_{\bar{b}\bar{c}}(\bar{X}, \bar{Y}) &\stackrel{\text{def}}{\equiv} P(\bar{X}, \bar{Y}) \wedge \neg Q_h(\bar{X}_h, \bar{b}, \bar{c}) \wedge \textstyle\bigwedge_{i \in L_h^+, i \neq h} \neg Q_i(\bar{X}_i, \bar{Y}_i, \bar{c}_i) \\
\xi^I_{\bar{b}\bar{c}}(\bar{X}, \bar{Y}, \bar{Z}_I) &\stackrel{\text{def}}{\equiv} (\forall \bar{X}')[(\psi_{\bar{b}\bar{c}}(\bar{X}', \bar{Y}) \wedge \bar{X}' =_M \bar{X}) \Rightarrow \textstyle\bigvee_{i \in I} Q_i(\bar{X}'_i, \bar{Y}_i, \bar{Z}_i)] \\
& \quad \wedge [\bar{Z}_I =_{I,h} \bar{c}]
\end{aligned}$$

Intuitively, each tuple $(\bar{X}, \bar{Y}, \bar{Z})$ in $P(\bar{X}, \bar{Y}) \wedge R(\bar{Y}, \bar{Z})$ that uses $Q_h(\bar{a}, \bar{b}, \bar{c})$ as a cover, must have some alternative cover. The second line in (12) essentially provides cover for these tuples. However, the Z-values, especially for those variables not in $\bar{Z}_h$, must be bounded. Intuitively, the first line in (12) essentially provides the boundability conditions, and the $\xi^I_{\bar{b}\bar{c}}$'s define the bounds (analogous to (10) in the extended example). As with the test for insertion, all nonempty subsets $I \subseteq L_h^-$ need to be considered here to make sure that $\xi^I_{\bar{b}\bar{c}}$ can be finitely evaluated.

Simplifications. When the index set over which a conjunction or a disjunction ranges becomes empty, the conjunction (resp disjunction) can be treated as vacuous truth (resp falsity), enabling test expressions to simplify. For instance, when there are no negated local subgoals, the complete test for safe insertion simplifies to $(\exists \bar{X})[P(\bar{X}, \bar{b}) \wedge \bar{X} =_M \bar{a}]$. Another source of complexity in the test expressions was introduced in order to make their evaluation safe. However, by carefully factoring out the Q_i's that use no Z-variables, and by grouping together those that use the same subset of Z-variables, drastic simplifications can be achieved. For instance, when there are no negated remote subgoals and all the negated local subgoals use the same Z-variables (say $\bar{Z}$), the insertion test degenerates to $(\exists \bar{X}) P(\bar{X}, \bar{b}) \wedge (\forall \bar{Z})(\phi_{\bar{b}}(\bar{Z}) \Rightarrow \bigvee_{i \in L} Q_i(\bar{a}_i, \bar{b}_i, \bar{Z}))$ where $\phi_{\bar{b}}(\bar{Z})$ is defined to be $(\exists \bar{X})[P(\bar{X}, \bar{b}) \wedge \bigvee_{i \in L} Q_i(\bar{X}_i, \bar{b}_i, \bar{Z})] \wedge (\forall \bar{X})[P(\bar{X}, \bar{b}) \Rightarrow \bigvee_{i \in L} Q_i(\bar{X}_i, \bar{b}_i, \bar{Z})]$. Thus, there are nontrivial classes of constraints whose CLT can be generated in time polynomial in the size of the constraints. More opportunities for simplification are explained in [H96atr].

4.4 Beyond CQC$^\neg$ constraints

The results we just presented are not limited to CQC$^\neg$ constraints as defined in (2). In fact, P and the local Q_i's may be *any query* not involving remote EDB's, as long as their updates can be computed from the EDB updates. This observation alone already allows us to extend the results to a considerably larger class of constraints. In addition, subqueries R and the remote Q_j's may be any query not involving local EDB's, as long as the Q_j's instances can be "arbitrarily realized" and R's instances can be "arbitrarily realized as singletons". We allow for instance $Q_j(\bar{X}_j, \bar{Y}_j, \bar{Z}_j)$ to actually represent $(\exists \bar{T}_j)\ q(\bar{X}_j, \bar{Y}_j, \bar{Z}_j, \bar{T}_j)$ for some remote predicate q, where the T-variables have been projected out. Thus, our complete test results can be used to enforce familiar data dependencies such as inclusion and tuple generating dependencies. We also allow for instance $R(\bar{Y}, \bar{Z})$ to represent the union of conjunctive queries $R_k(\bar{Y}, \bar{Z})$'s as long as no predicate occurs more than once in these queries. Thus, our results are also applicable to a limited form of union of CQC$^\neg$'s.

5 Sets of Updates

Section 4 gave CLT solutions under single updates. These solutions are now extended to insertion sets and deletion sets. Due to space limitation, we will just briefly mention the results. The formal results and their proofs can be found in [H96atr]. In the following, we assume $L = \{1, \ldots, m\}$ and $h \in L$.

- Insertions to positive subgoals are *independently testable*: the complete test under the insertion of $(\Delta P,\ \Delta Q_1, \ldots, \Delta Q_m)$ is the conjunction, for t ranging over ΔP, of CLT under the insertion of $(t,\ \Delta Q_1, \ldots, \Delta Q_m)$. The CLT's for the latter type of insertion updates generalize the results from previous section by replacing some occurrences of Q_i in the test expression with $Q_i \cup \Delta Q_i$.
- Deletions from the same negated subgoal are independently testable: the complete test under the deletion of $(\Delta P,\ \Delta Q_h)$ is the conjunction, for t ranging over ΔQ_h, of CLT under the deletion of $(\Delta P, t)$. The CLT's for the latter type of deletion updates generalize the results from previous section by replacing some occurrences of P in the test expression with $P - \Delta P$.
- Deletions from multiple negated subgoals are *testable in chain*: testing safety of deletion update $(\Delta Q_1, \ldots, \Delta Q_m)$ can be done by testing safety of deletion update (ΔQ_i) in the instance of the local relations that results from deleting $(\Delta Q_1, \ldots, \Delta Q_{i-1})$ from the original instance.

6 Conclusion

In this paper, we considered CQC$^\neg$'s for which, while an important class of constraints, the problem of finding efficient CLT's remained open. We solved the CLT problem for general CQC$^\neg$ constraints for single updates when remote predicates have single occurrences in the constraint queries. We gave complete test solutions in a form that can easily be implemented in SQL. The time to generate these tests is at most exponential in the size of the given constraint. We also pointed out opportunities for substantially reducing the complexity of these tests. Since all the tests can be implemented in nonrecursive Datalog$^\neg$, executing them takes time at most polynomial in the size of the local relations.

Throughout this paper, we have assumed single occurrences of the remote predicates in the constraint query. This restriction was needed to ensure that it is always possible to choose an instance of the remote EDB relations such that arbitrary answers of the subqueries R and Q_j's can be realized. If the restriction is ignored, while all tests remain sound, their completeness can no longer be guaranteed. Indeed, if we allow multiple occurrences of the remote predicates, new dependencies are created within and among the instances of R and Q_j's. This observation suggests that for a test to be complete, the dependencies, induced by the particular structure of the remote queries on their results, must be embodied in the test's condition. When repeated remote predicates are allowed but only within R, we show in [H96atr] how our techniques can be extended to derive CLT's. Essentially, under the insertion of $P(\bar{a}, \bar{b})$, instead of looking for tuples $(\bar{X}, \bar{Y})$ in P that satisfy $\bar{Y} = \bar{b}$ (among other conditions), the search is extended to tuples whose $\bar{Y}$ may now take any value from some "closure" of $R(\bar{b}, \bar{Z})$. This closure degenerates to $\{\bar{b}\}$ when R has no repeated predicates.

Acknowledgments. We thank Jeff Ullman and Serge Abiteboul for their valuable comments regarding both technical contents and presentation of this material.

References

[G94] Gupta A.: *Partial Information Based Integrity Constraint Checking.* PhD Thesis, Stanford University, November 1994.

[GB95] Gupta A. and Blakeley J. A.: Using Partial Information to Update Materialized Views. In *Information Systems,* 20(8), pp. 641–662, 1995.

[GSUW94] Gupta A., Sagiv Y., Ullman J. D. and Widom J.: Constraint Checking with Partial Information. *Proc. 13th ACM Symp. on PODS,* 1994, pp. 45–55.

[H96atr] Huyn N.: Testing CQC$^\neg$ constraints under limited data access. *Unpublished Technical Report* available as URL `http://www-db.stanford.edu/pub/papers/cqcnclt-tr.ps`.

[H96b] Huyn N.: Efficient View Self-Maintenance. *Proc. ACM Workshop on Materialized Views,* 1996, pp. 17–25.

[LS93] Levy A. and Sagiv Y.: Queries Independent of Updates. *Proc. 19th International Conf. on Very Large Data Bases,* 1993, pp. 171–181.

[TH93] Tiwari S. and Howard H. C.: Constraint Management on Distributed AEC Databases. *Fifth International Conf. on Computing in Civil and Building Engineering,* ASCE, 1993, pp. 1147–1154.

[TB88] Tompa F. W. and Blakeley J. A.: Maintaining Materialized Views Without Accessing Base Data. In *Information Systems,* 13(4), pp. 393–406, 1988.

[Ull88] Ullman J. D.: *Principles of Database and Knowledge-Base Systems,* Volumes 1 and 2. Computer Science Press, 1988.

This article was processed using the Lscaled 913ATEX macro package with LLNCS style