Testing CQC$^\neg$ constraints under limited data access

Nam Huyn
Department of Computer Science
Stanford University
Stanford, California, 94305
huyn@cs.stanford.edu

Abstract

In many important application areas such as collaborative design, mobile computing and enterprise information systems, total data accessibility can not be assumed. Even if all data can be accessed, some of them may incur such a high cost that their access should only be considered as a last resort. Efficiently checking if a local update violates a global integrity constraint spanning several relations poses new challenges when not all these relations are to be looked at. Without being given access to all relations under constraint, 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 accessible relations can in principle be used to infer something about the inaccessible relations. Under limited data accessibility, the "complete local test" (or CLT in short) represents the best strategy to evaluate if the update can potentially violate the constraint.

This paper addresses the problem of finding CLT's for an important class of constraints that are very common in practice: constraints expressible as conjunctive queries with negated subgoals (abbreviated CQC$^\neg$.) We show that for single updates, all CQC$^\neg$ constraints admit a CLT that can be expressed in nonrecursive Datalog$^\neg$ when each inaccessible relation has a single occurrence in the constraint query. We then extend this result to certain sets of updates. The significance of our results is not only that complete tests for CQC$^\neg$ constraints can be generated and executed efficiently, but that they can also be executed on traditional query-evaluation engines.

1 Introduction

We consider the problem of incrementally checking global integrity constraints when not all the relations under constraint are accessible. That is, given an update to the relations and a constraint to be maintained, we would like to decide whether or not the update causes new violation of the constraint, looking only at accessible relations.

Partial Information

In many important global information systems, total accessibility can not be assumed. In some cases, information access may be restricted for security reasons. In other cases, remote information systems may be disconnected temporarily due to mobility requirements. Cost may also be a factor in deciding not to look at certain information sources if not absolutely needed. Finally, partial access can be used as an approach to optimize constraint maintenance in a distributed environment by first performing local checks for potential constraint violations before considering more expensive global checks.

When partial accessibility must be factored into the problem of constraint checking, we face new challenges: without looking at all the relations under constraint, how can one meaningfully and efficiently check a constraint for violation? When the constraint is known to be satisfied prior to the update, the state of the accessible relations can, in principle, be used to infer about the possible states the inaccessible relations can be in. This observation is the basis for the existence of tests that can still perform adequately despite the lack of information.

Complete Tests

Test procedures we are interested in must be sound, that is, never miss any violation. While soundness is an absolute requirement to assure integrity of the overall information system, it alone is not sufficient to make them useful: a degenerate test procedure that always results in predicting potential violation is certainly sound but too conservative to be of practical use. Thus, we require test procedures to be complete as well: given an update, when the test predicts violation, there is always a state of the inaccessible relations that satisfies the constraint before the update but violates it after. In the remainder of this paper, we will consider only test procedures that have these properties. Such a test will be called *Complete Local Test* or CLT in short. For a thorough introduction to the concept of CLT, refer to [GSUW94].

EXAMPLE 1.1 Consider a large distributed system where service accesses are under security control. In this simplified access control model:

- Each user is assigned exactly one clearance level. Predicate *user(Uid,Clearance)* represents this relationship.
- Each service belongs to at most one class. Predicate *service(Sid,Sclass)* represents this relationship.
- All service classes whose access is allowed for a given clearance level are explicitly specified by predicate *auth(Clearance,Sclass)*.
- All service accesses are logged, as represented by predicate *access(Uid,Sid)*.

Violation of the security policy can be captured by the condition

$$(\exists U)(\exists C)(\exists S)(\exists Sc)[user(U,C) \wedge access(U,S) \wedge service(S,Sc) \wedge \neg auth(C,Sc)] \qquad (1)$$

stating that a user is authorized to access a service only if he has the proper clearance for the corresponding class. In order to enforce the policy, every access request is checked by the system (on behalf of the user) to ascertain that the access is legitimate.

Now suppose user *joe* wants to use service *vipinfo* and assume that it has already been established that *joe* has the *confid* clearance.

- If both relations *service* and *auth* are inaccessible, it is not immediately obvious if anything can be done to validate *joe*'s access since the meaning of clearance and service class is essentially captured by *service* and *auth*. Looking deeper, we observe that if we can find some users (not necessarily *joe*) with the same clearance as *joe* and having successful access to *vipinfo*, then *joe*'s access ought to be honored. In fact, no matter how clearance and service class are defined, there is no way to discriminate between *joe* and these other users. Conversely, if no such users can be found, there are cases where *joe*'s access to *vipinfo* is illegal. Let CV denote the clearance of all users who have successfully accessed *vipinfo*. We know that *confid* is not in CV. Indeed, *auth* could have been defined to authorize only the levels in CV to access *vipinfo*'s class and *service* to specify the class of only *vipinfo*, and *joe*'s access would be clearly illegal.

- If only relation *service* is inaccessible, the previous test can still be used to validate *joe*'s access to *vipinfo*. What is less obvious is whether it is the only validation test. Intuitively, having more information available (namely *auth*) should help us do better. Indeed, consider the situation where the levels in CV are actually defined to be "lower" than *confid*: *confid* is authorized to access any service class that all levels in CV are cleared to access in common. This situation defines a different validation test that takes advantage of the availability of *auth*. It turns out this test is the best that can be done to validate *joe*'s access to *vipinfo* not looking at the *service* relation.

▮

Constraints With Negation

Constraints such as (1) are very common in practice. We call them *Conjunctive-Query Constraints with Negation*, or $\text{CQC}^\neg$ in short. With $\text{CQC}^\neg$'s, allowable combinations of tuples from some relations can be explicitly specified (either totally or partially) in some other relations and those that are not specified are construed to be illegal. The more familiar data dependencies such inclusion and tuple generating dependencies are very specialized subsets of $\text{CQC}^\neg$'s. This paper mainly considers the problem of finding CLT's for constraints involving negation in general and for $\text{CQC}^\neg$'s in particular. The $\text{CQC}^\neg$ class is shown in Figure 1 as the shaded oval. To the best of our knowledge, this problem is still open. Many questions that have practical significance are not immediately obvious. Do $\text{CQC}^\neg$'s admit test procedures that are complete and that can be efficiently generated? Do $\text{CQC}^\neg$'s admit CLT's that can be efficiently executed, i.e., in time polynomial in the size of the accessible relations? Can these CLT's be expressed as queries in conventional query languages so that they can be executed using conventional query-evaluation engines? In this paper, we establish the following results:

- For single updates to accessible relations, all $\text{CQC}^\neg$'s admit complete local tests that can be expressed as nonrecursive Datalog queries with negation, when each inaccessible predicate has single occurrence in the constraint query (shown as a dashed arc in Figure 1). Using different techniques we recently developed but not described in this paper, our results are extended to cover the entire CQ class.

- These results are extended to arbitrary sets of insertion updates and to certain sets of deletion updates to accessible relations.

- All solution tests can be generated in time at most exponential in the size of the constraints. The complete tests are given in a form that makes their expression in $\text{Datalog}^\neg$ (and other traditional query languages) straightforward.

Related Work

The notions of local tests and completeness were first introduced in [GW93] and [Gupta94], [GSUW94] respectively, and were embodied in an actual system supporting collaborative design in building construction [Tiwari93]. The most powerful result for CLT's that can be found in [Gupta94] 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 Figure 1). Negation was considered in [GSUW94] but only for local tests that are not complete. Also our approach has the advantage that the CLT conditions can be *directly* implemented as queries while in [Gupta94] the containment tests used need to be specialized to the particular case.

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.

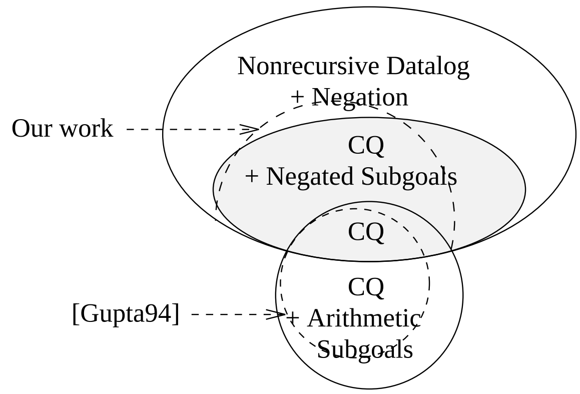

Figure 1: Constraint classes for the CLT problem

Outline

The rest of this paper is organized as follows. In the next section, we provide some basic concepts on maintaining CQC¬ 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 details in deriving and validating complete tests for our running example. In the following three sections, we present the main results of this paper. Finally in Section 7, we summarize our work and discuss possible extensions which are elaborated in the Appendix.

2 Preliminaries

Datalog¬

In this paper, integrity constraints (and their complete tests if any) are modeled as queries expressed by Datalog¬ programs. A Datalog¬ 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¬ 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¬ 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¬), that is, Datalog¬ 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 predicates are inaccessible (aka remote) and other accessible (aka local), we introduce a notation that makes the remote vs. local distinction explicit. This notation also makes variable use in subgoals explicit. Thus, for the remainder of this paper, CQC$^\neg$'s are represented as

$$panic :- \underbrace{P(X,Y) \ \& \ \bigwedge_{i \in L} \neg Q_i(X_i, Y_i, Z_i)}_{local} \ \& \ \underbrace{\bigwedge_{j \in M} \neg Q_j(X_j, Y_j, Z_j) \ \& \ R(Y,Z)}_{remote}. \qquad (2)$$

where

- $P(X,Y)$ denotes a conjunction of zero or more local positive subgoals (i.e., subgoals with local predicates), X denote the set of variables that are used in positive local subgoals but not in positive remote subgoals, Y denote the set of variables used in both positive local and remote subgoals. Thus, X and Y are mutually disjoint.
- $R(Y,Z)$ denotes a conjunction of zero or more remote positive subgoals (i.e., subgoals with remote predicates), Z denote the set of variables that are used in positive remote subgoals but not in positive local subgoals, Y denote the set of variables used in both positive local and remote subgoals. Thus X,Y,Z are mutually disjoint.
- The set $\{1, \ldots, n\}$, where n is the number of negated subgoals, is partitioned into two subsets L and M: L indexes the negated local subgoals, that is, for $i \in L$, $Q_i(X_i, Y_i, Z_i)$ denotes a subgoal with some local predicate; M indexes the negated remote subgoals, that is, for $j \in M$, $Q_j(X_j, Y_j, Z_j)$ denotes a subgoal with some remote predicate.
- $X_i \subseteq X$, $Y_i \subseteq Y$, $Z_i \subseteq Z$ are assumed for all $i \in \{1, \ldots, n\}$ as required for rule safeness.

EXAMPLE 2.1 Constraint (1) in Example 1.1 is represented as:

$$panic :- user(U,C) \ \& \ access(U,S) \ \& \ service(S,Sc) \ \& \ \neg auth(C,Sc).$$

When only EDB predicate *service* is not accessible, the constraint can be rewritten in notation (2) where P denotes the conjunction $user(U,C) \wedge access(U,S)$, R denotes the subgoal $service(S,Sc)$, X denotes $\{U,C\}$, Y denotes $\{S\}$, Z denotes $\{Sc\}$, Q_1 denotes the negated subgoal $auth(C,Sc)$, X_1 denotes $\{C\}$, $Y_1 = \{\}$, Z_1 denotes $\{Sc\}$. ∎

Updates

Given a constraint and accessibility information about the EDB predicates as in (2), we only consider insertion and deletion updates to the EDB relations that are accessible. More precisely, let ΔP be a set of tuples with P's arity and for each accessible Q_i, let ΔQ_i be a set of tuples with Q_i's arity. Insertion and deletion updates are represented by their net effects on P and the Q_i's. Thus, insertion update ΔP represents the tuples that are effectively added to P. Deletion update ΔP represents the tuples that are effectively removed from P, that is, all tuples in ΔP are in P before the update and none of them remains in P after the update. The same convention applies to the accessible Q_i's. ΔP and the ΔQ_i's are called *delta sets.*

In the rest of this paper, we assume that ΔP and ΔQ_i's are given. If we are only given updates to the EDB relations, we would like to point out that there are many ways to determine the delta sets from these primitive updates. To compute ΔP for instance, the easiest way is to compare the "before image" and "after image" of P's results, and the difference can be captured as an insertion ΔP and a deletion ΔP. There are other algebraic methods that can more efficiently determine delta sets from the updates to the underlying EDB relations. These algebraic methods are fairly straightforward especially when P and the Q_i's are conjunctive queries over the EDB relations (see Appendix for more details).

Tests

Given a set of updates to the accessible EDB relations, we are interested in finding complete tests whose inputs are limited to P, ΔP, the accessible Q_i's and the ΔQ_i's. For all practical purposes, these tests will be used to incrementally check whether global consistency is still maintained after each batch of updates. We shall be looking for complete tests that are expressible as first order queries over the accessible EDB relations and the delta sets. These tests will be implemented as nonrecursive Datalog$^\neg$ programs. In the remainder of this paper, the terms "complete test", "complete local test" and "CLT" will be used interchangeably.

3 Detailed example

In this section, we will go through the process of finding a complete local test for our running example. We use an example to informally explain our approach for finding CLTs and to illustrate the main elements used in showing that a given test is both sound and complete. Recall from Example 1.1 the problem of finding complete test for constraint

$$panic \text{ :- } \underbrace{user(U,C)\ \&\ access(U,S)\ \&\ \neg auth(C,Sc)}_{local}\ \&\ \underbrace{service(S,Sc)}_{remote}$$

when the *service* relation is not accessible. We first consider a simple insertion and later on a simple deletion.

Characterizing when a simple insertion is safe

Suppose we want to insert *access(joe,vipinfo)*. To make the solution nontrivial, assume that relation *user* has some tuple (say *user(joe,confid)*) that joins with *access(joe,vipinfo)* (otherwise the insertion can never cause new violation). In order to assure no new violation results from the insertion, whatever class *Sc* service *vipinfo* belongs to, *joe*'s *confid* clearance must have been authorized for *Sc*, that is

$$(\forall Sc)[service(vipinfo,Sc) \Rightarrow auth(confid,Sc)] \tag{3}$$

Using adversary arguments

The problem is that relation *service* is not accessible. From our point of view, we should be prepared for the worst case where the adversary (relation *service* in our example) deliberately attempts to choose a state that satisfies the constraint prior to the insertion but violates it after. For instance, if the adversary finds out no one ever accessed *vipinfo* before, it will be able to successfully trick us by simply inventing a new class for *vipinfo* that does not appear in *auth*. So the following condition is necessarily part of our test:

$$(\exists UC)[user(U,C)\ \&\ access(U,vipinfo)] \tag{4}$$

By the same token, how can we guarantee condition (3) against all possible moves the adversary can take? A key observation is that in order to assure constraint satisfaction prior to the insertion, *the adversary is not totally free to choose its state.* If we can put a bound on all potential states the adversary can possibly assume, we can effectively come up with a test that guarantees the insertion is safe no matter the adversary does.

Characterizing inaccessible relations

We mentioned earlier that in order to put any bound on *service*, condition (4) must be true. So assume there have been previous successful accesses to *vipinfo* (i.e. condition (4)). In order to assure validity of all these previous accesses, the adversary cannot choose a class *Sc* for service *vipinfo* for which some previous successful access to *vipinfo* has not been explicitly authorized. In order words, not only *Sc* must occur in relation *auth*, but the clearance levels in all previous successful accesses to *vipinfo* must also be authorized for *Sc*. So formally:

$$(\forall Sc)[service(vipinfo, Sc) \Rightarrow quotient(Sc)] \tag{5}$$

where $quotient(Sc)$, defined as

$$(\forall UC)[(user(U,C) \wedge access(U, vipinfo)) \Rightarrow auth(C, Sc)] \tag{6}$$

essentially puts a bound on all possible values of *Sc* our adversary can choose. Is that bound reachable? It turns out that under condition (4), condition (5) is both sufficient and necessary to guarantee no illegal access to *vipinfo*.

Lemma 3.1 *There is no illegal access to vipinfo if and only if the following condition holds:*

$$[(\exists UC)user(U,C) \wedge access(U, vipinfo)] \Rightarrow (\forall Sc)[service(vipinfo, Sc) \Rightarrow quotient(Sc)] \tag{7}$$

∎

To verify sufficiency, assume (7) holds and let *u,c,sc* be arbitrary constants such that *user(u,c)*, *access(u,vipinfo)* and *service(vipinfo,sc)* hold. We need to show that *auth(c,sc)* also holds. Substituting *u,c,sc* into (7), we can deduce *quotient(sc)*. Substituting *u,c* into the definition of *quotient(sc)*, we deduce *auth(c,sc)*.

To verify necessity, assume no illegal access to *vipinfo* and let *u,c,sc* be arbitrary constants that satisfy the premises of both implications in (7): that is, *user(u,c)*, *access(u,vipinfo)* and *service(vipinfo,sc)* hold. We need to show *quotient(sc)* also holds. So let *u',c'* arbitrary constants such that *user(u',c')*, *access(u',vipinfo)* hold. It remains to show *auth(c',sc)* also holds. Now by hypothesis, user *u'* access to *vipinfo* is not illegal. Therefore *auth(c',sc)* must be true.

Putting it altogether

By now, we have all the ingredients to build a test. We already mentioned that condition (4) must be a consequence of the test. Also, we need another condition from which (3) can be guaranteed for all possible states of *service* that satisfy (7).

Consider the following condition:

$$[(\exists UC)user(U,C) \wedge access(U, vipinfo)] \wedge (\forall Sc)[quotient(Sc) \Rightarrow auth(confid, Sc)] \tag{8}$$

Proposition 3.1 *Condition (8) is a complete local test that the insertion of access(joe,vipinfo) is safe.*
∎

To show soundness, assume (8) is satisfied and no violation existed prior to the insertion. In order to show no violation is introduced by the insertion, let *service(vipinfo,sc)* hold for some constant *sc*. We need to show *auth(confid,sc)* also holds. Since there is no prior violation, by applying Lemma 3.1 and then substituting *sc* into 7, we can infer *quotient(sc)*. By again substituting *sc* into the second subcondition in (8), we infer *auth(confid,sc)*.

To show completeness, assume (8) is not satisfied. Either the first subcondition in (8) is false or the second is false (but the first is true). In either case, we need to show we can choose a instance of relation *service* that satisfies the constraint before the insertion but violates it after. In the first case, *service* consists of a single tuple <*vipinfo,sc*> where *sc* is a new constant that does not occur in *auth*. Clearly no violation pre-existed since falsity of the first subcondition in (8) assures there is no pre-existing access to *vipinfo*. Also, the insertion causes the constraint to be violated since relation *auth* cannot contain tuple <*confid,sc*> by construction of *sc*. In the second case, there is a value *sc* that falsifies the second subcondition in (8), that is, *quotient(sc)* is true but *auth(confid,sc)* is false. Again, choose *service* to consist of the single tuple <*vipinfo,sc*>. First, the insertion causes a violation since *auth(confid,sc)* is false. Also, to show there is no pre-existing violation, we use Lemma 3.1. Since the premise in (7) is true by hypothesis and since *service* has only one tuple (namely <*vipinfo,sc*>), we only need to verify *quotient(sc)* holds. Indeed *quotient(sc)* is true by selection of *sc*.

Testing safe insertion in Datalog$^{\neg}$

Condition (8) would be easy to implement as a Datalog$^{\neg}$ program (or as a query in traditional query languages) if we know how to *finitely* evaluate *quotient(Sc)*. The problem is that formula (6), the current definition of *quotient(Sc)*, is not safe. For instance, if there is no values for U,C that satisfies *user(U,C)* $\wedge$ *access(U,vipinfo)*, any value for *Sc* would satisfy *quotient(Sc)*. Fortunately, in the context of our test, this situation has been factored out, and *quotient(Sc)* can be redefined to be safe as follows, without changing the validity of the results (the proof is omitted here):

$$\begin{array}{rl} (\exists UC) & [\, user(U,C) \wedge access(U, vipinfo) \wedge auth(C, Sc)] \\ \wedge(\forall UC) & [\, (user(U,C) \wedge access(U, vipinfo)) \Rightarrow auth(C, Sc)] \end{array}$$

Finally, assuming that *joe* is known to be assigned clearance *confid*, a Datalog$^{\neg}$ program that implements the complete test (8) for safely inserting *access(joe,vipinfo)* is shown (with *safeinsert* as the query predicate):

$$\begin{array}{rcl} p(U,C) & :- & user(U,C)\ \&\ access(U, vipinfo). \\ local(Sc) & :- & p(U,C)\ \&\ auth(C, Sc). \\ forbid(Sc) & :- & local(Sc)\ \&\ p(U,C)\ \&\ \neg auth(C, Sc). \\ quotient(Sc) & :- & local(Sc)\ \&\ \neg forbid(Sc). \\ notcovered & :- & quotient(Sc)\ \&\ \neg auth(confid, Sc). \\ safeinsert & :- & p(U,C)\ \&\ \neg notcovered. \end{array}$$

We now turn to the problem for a simple deletion.

Simple deletion

Suppose we want to delete *auth(confid,secured)*. For the deletion to be safe, we want to avoid situations where the tuple to be deleted is being used as the sole way to legitimize the existence of certain tuples in other relations. For instance, the tuple <*confid,secured*> in relation *auth* could actually be used to provide "cover" for some user with the *confid* clearance accessing some service belonging to the *secured* class. In this case, if we remove the cover, there would not be any other tuple (presumably from another negated subgoal) that can provide the "same" cover. Clearly such situations must be disallowed, that is

$$(\forall US)[(user(U, confid) \wedge access(U,S)) \Rightarrow \neg service(S, secured)] \tag{9}$$

Since we assume our constraint is satisfied prior to the deletion, the idea of putting a bound on the adversary relation *service* applies here as well. This bound is stated in the following lemma:

Lemma 3.2 *There is no illegal access to any service of class "secured" if and only if the following condition holds:*

$$(\forall S)[service(S, secured) \Rightarrow quotient'(S)] \tag{10}$$

where $quotient'(S)$ is defined as

$$(\forall UC)[(user(U,C) \wedge access(U,S)) \Rightarrow auth(C, secured)] \tag{11}$$

∎

The proof is very similar to the proof of Lemma 3.1 and is omitted here.

By "eliminating" relation *service* from (9) using (10), we obtain the following condition

$$(\forall US)[(user(U, confid) \wedge access(U,S)) \Rightarrow \neg quotient'(S)] \tag{12}$$

which we claim to be sound and complete in the following Proposition.

Proposition 3.2 *Condition (12) is a complete local test that the deletion of auth(confid,secured) is safe.* ∎

To show soundness, assume (12) is satisfied and no violation existed prior to the deletion. We will show by contradiction that no violation is introduced by the deletion. So assume that deletion of *auth(confid,secured)* causes violation, i.e., there is a user *u* and a service *s* that satisfy *user(u,confid)*, *access(u,s)* and *service(s,secured)*. By substituting *u,s* into (12), we infer *quotient'(s)* to be false. By definition of *quotient'(s)*, there must be some constants *u',c'* such that *user(u',c')* and *access(u',s)* hold but *auth(c',secured)* is false. Since *service(s,secured)* is true, we have a violation prior to the deletion.

To show completeness, assume (12) is not satisfied. There must be constants *u,s* that falsify (12): *user(u,confid)*, *access(u,s)* and *quotient'(s)* all hold. Consider an instance of relation *service* that consists of the sole tuple <*s,secured*>. Using Lemma 3.2, we can easily verify that with this instance of *service*, there is no prior violation. After <*confid,secured*> has been deleted from *auth*, *user(u,confid)*, *access(u,s)* and *service(s,secured)* are clearly in violation.

Testing safe deletion in Datalog¬

A Datalog¬ program that implements the complete test (12) for safely deleting *auth(confid,secured)* is shown (with $safedelete$ as the query predicate):

$$\begin{array}{rcl} p'(U,C,S) & :- & user(U,C)\ \&\ access(U,S). \\ forbid'(S) & :- & p'(U,C,S)\ \&\ \neg auth(C, secured). \\ quotient'(S) & :- & p'(U,C,S)\ \&\ \neg forbid'(S). \\ unsafe & :- & p'(U, confid, S)\ \&\ quotient'(S). \\ safedelete & :- & \neg unsafe. \end{array}$$

4 Single insertions

4.1 No negated subgoals

This section examines contraints of the form

$$panic :- \underbrace{P(X,Y)}_{local}\ \&\ \underbrace{R(Y,Z)}_{remote}. \tag{13}$$

Theorem 4.1 *The following condition is a complete local test that the insertion of P(a,b) is safe:*

$$(\exists X)P(X,b) \tag{14}$$

▮

Proof: To show soundness, assume test condition (14) is satisfied, that is, there is some value a' such that satisfies $P(a',b)$. Since there is no violation before the insertion, there is no value of Z that satisfies $R(b,Z)$. Therefore, $P(a,b)$ cannot be in violation of the constraint.

To show completeness, assume (14) is false. We need to find an instance of R that satisfies the constraint before the insertion but violates it after. Let us choose $R = \{< b, c >\}$ where c is some arbitrary constant. There cannot be any constraint violation prior to the insertion since no tuple in P joins with R (by hypothesis). After the insertion, $P(a,b)$ and $R(b,c)$ clearly violates the constraint. ▮

4.2 Single negated subgoals with local predicate

This section examines contraints of the form

$$panic :- \underbrace{P(X,Y) \ \& \ \neg Q(X,Y_1,Z)}_{local} \ \& \ \underbrace{R(Y,Z)}_{remote}. \tag{15}$$

Note that generality is not lost by assuming that P and Q share exactly the same X-variables, and Q and R the same Z-variables. In fact, if there are variables strictly local to P (resp. R), we can always consider new queries by projecting them out of P (resp. R). Also assume $Z \neq \emptyset$.

Definition 4.1 (Simple Quotient): Let us define the simple quotient Ψ_{QP} by

$$\Psi_{QP}(Y,Z) \stackrel{\text{def}}{\equiv} (\exists X)[P(X,Y) \wedge Q(X,Y_1,Z)] \wedge (\forall X)[P(X,Y) \Rightarrow Q(X,Y_1,Z)]^1 \tag{16}$$

▮

When there is no contraint violation, the following lemma gives a necessary and sufficient characterization of R, based on local states.

Lemma 4.1 *The constraint is satisfied if and only if the following condition holds:*

$$(\forall YZ)[R(Y,Z) \Rightarrow [(\forall X)\neg P(X,Y)] \vee \Psi_{QP}(Y,Z)] \tag{17}$$

▮

Proof:

From the definition of the constraint (15), it is easy to see that a necessary and sufficient condition for constraint satisfaction is given by

$$(\forall XYZ)[(P(X,Y) \wedge R(Y,Z)) \Rightarrow Q(X,Y_1,Z)] \tag{18}$$

We want to show that (17) is equivalent to this condition.

[1] We need to clarify the notation used here and throughout the rest. Variables used in formulas, such as Y and Y_1, are actually meta-variables that denotes sets of variables. What could be confusing is that some meta-variables (e.g. Y_1) appear to be free but actually do denote sets of variables that are indirectly bound (e.g. through Y.)

To show sufficiency, assume that (17) holds and let a, b and c be values for X, Y and Z that make the premise in (18) true. That is, $P(a, b)$ and $R(b, c)$ hold. We need to show that $Q(a, b_1, c)$ [2] also holds. Since both $(\exists X)P(X, b)$ and $R(b, c)$ hold, the only way that (17) can hold is that $\Psi_{QP}(b, c)$ holds. Using the definition (16) of $\Psi_{QP}(b, c)$, it follows that $P(X, b) \Rightarrow Q(X, b_1, c)$ is true for all X, and for $X = a$ in particular. But $P(a, b)$ holds. Therefore $Q(a, b_1, c)$ holds.

To show necessity, assume that (18) holds and let b and c be values for Y and Z that make the premise in (17) true, that is, $R(b, c)$ hold. We need to show that either $(\forall X)\neg P(X, b)$ or $\Psi_{QP}(b, c)$ holds. So assume that there is a value a for X that makes $P(a, b)$ true. Using (18), we can infer that $Q(a, b_1, c)$ holds. We can verify that the first conjunct in the formula for $\Psi_{QP}(b, c)$ is satisfied. The second conjunct in $\Psi_{QP}(b, c)$ can be shown to be satisfied in a very similar way. That is, assume that there is a value a' for X that makes $P(X, b)$ true. To show that $Q(a', b_1, c)$ holds, we use (18) again. Therefore, $\Psi_{QP}(b, c)$ holds. ∎

Theorem 4.2 *The following condition is a complete local test that the insertion of P(a,b) is safe:*

$$[(\exists X)P(X, b)] \wedge (\forall Z)[\Psi_{QP}(b, Z) \Rightarrow Q(a, b_1, Z)] \tag{19}$$

∎

Proof:

To show soundness, assume the test (19) is satisfied. We need to show that $P(a, b)$ does not introduce any new violation, that is, for any Z that makes $R(b, Z)$ true, $Q(a, b_1, Z)$ must hold. Since there is no violation before the insertion and since $(\exists X)P(X, b)$ holds by hypothesis, applying Lemma 4.1 for $Y = b$ tells us that $(\forall Z)R(b, Z) \Rightarrow \Psi_{QP}(b, Z)$ holds. Using the the second conjunct in (19), we infer that $(\forall Z)R(b, Z) \Rightarrow Q(a, b_1, Z)$ holds.

To show completeness, assume the test (19) is not satisfied. Then, either $(\forall X)\neg P(X, b)$ or the second conjunct in (19) is false. In either case, we need to show that we can choose an R that satisfies the constraint before the insertion but that violates it after. In the first case, all we need is to pick a value c for Z that does not appear in Q and choose $R = \{< b, c >\}$. Clearly no violation existed before the insertion since there was no tuple in P that joins with it. Also, inserting $P(a, b)$ causes a violation since $Q(a, b_1, c)$ is false (c does not appear in Q.) In the second case, there is a value c such that $\Psi_{QP}(b, c)$ is true but $Q(a, b_1, c)$ false. Choose $R = \{< b, c >\}$. A violation is clearly created by inserting $P(a, b)$, since $Q(a, b_1, c)$ is false in this case. Also we claim that no violation existed before. To show this, we will use Lemma 4.1 that requires showing that either $(\forall X)\neg P(X, b)$ or $\Psi_{QP}(b, c)$ holds. But we already know that $(\forall X)\neg P(X, b)$ is false in this case and that $\Psi_{QP}(b, c)$ is true by hypothesis. ∎

Implementing the test

We show a Datalog program that implements the test that the insertion of $P(a, b)$ is safe, where *safeinsert* is the query predicate:

$$\begin{array}{rcl}
local(Z) & \text{:-} & P(X, b)\ \&\ Q(X, b_1, Z). \\
forbid(Z) & \text{:-} & local(Z)\ \&\ P(X, b)\ \&\ \neg Q(X, b_1, Z). \\
quotient(Z) & \text{:-} & local(Z)\ \&\ \neg forbid(Z). \\
notcovered & \text{:-} & quotient(Z)\ \&\ \neg Q(a, b_1, Z). \\
safeinsert & \text{:-} & P(X, b)\ \&\ \neg notcovered.
\end{array}$$

[2]Another notation to be clarified: b_1 denotes the projection of b onto the variables denoted by Y_1.

4.3 No negated remote subgoal

We now generalize the previous result to the case where several negated subgoals are allowed. This is also the general case where $M = \emptyset$. Constraints have the form

$$panic :- \underbrace{P(X,Y) \;\&\; \bigwedge_{i \in L} \neg Q_i(X_i, Y_i, Z_i)}_{local} \;\&\; \underbrace{R(Y,Z)}_{remote}. \tag{20}$$

Difficulties

At first, there appear to be many sources of difficulty when going from a single negation to multiple negations.

- Every attempt to reduce the conjunction of negated subgoals in (20) to a single negated subgoal as in (15) proves fruitless, especially when the variable sets X_i, Y_i and Z_i are allowed to change from subgoal to subgoal.
- In the case of single negation, we mentioned that variables that are strictly local to P can be eliminated by projecting them out of P. In the case of multiple negations, we can no longer make this simplification since the notion of "strictly local" is now relative to each negated subgoal.
- Variables local to R present a similar problem. However, as we will show later, this problem turns out to be worse since it requires a much bigger jump in the generalization process than the problem with variables local to P does.

Having said that, we now present a solution that requires generalizing the notion of quotient.

Definition 4.2 (Generalized Quotient): Let I be a non empty subset of $L = \{1, \ldots, n\}$. The generalized quotient Φ_{IP} is defined by

$$\Phi_{IP}(Y, Z_I) \stackrel{\text{def}}{\equiv} [\bigwedge_{i \in I} (\exists X)(P(X,Y) \wedge Q_i(X_i, Y_i, Z_i))] \wedge (\forall X)[P(X,Y) \Rightarrow \bigvee_{i \in I} Q_i(X_i, Y_i, Z_i)]^3 \tag{21}$$

∎

Note that the following Lemma and Theorem remain valid if we remove the first conjunct from the definition of $\Phi_{IP}(Y, Z_I)$ in (21.) The sole reason the first conjunct is included is to make sure the evaluation of $\Phi_{IP}(Y, Z_I)$ be safe.

Lemma 4.2 *There is no violation if and only if the following condition holds:*

$$(\forall YZ)[R(Y,Z) \Rightarrow [(\forall X)\neg P(X,Y)] \vee \bigvee_{I \subseteq L, I \neq \emptyset} \Phi_{IP}(Y, Z_I)] \tag{22}$$

∎

Proof: The structure of the proof is similar to the case of single negation. From the definition of the constraint (20), it is easy to see that an equivalent characterization of no violation is given by:

$$(\forall XYZ)[(P(X,Y) \wedge R(Y,Z)) \Rightarrow \bigvee_{i \in L} Q_i(X_i, Y_i, Z_i)] \tag{23}$$

[3] Another notation clarified: given a set of integers I, Z_I denotes the union of all variables from Z_i for all $i \in I$.

We want to show that (22) is both sufficient and necessary for this condition to hold.

To show sufficiency, assume that (22) holds and let a, b and c be values for X, Y and Z that make the premise in (23) true. That is, $P(a,b)$ and $R(b,c)$ hold. We need to show that $\bigvee_{i \in L} Q_i(a_i, b_i, c_i)$ also holds. Since both $(\exists X)P(X,b)$ and $R(b,c)$ hold, the only way that (22) can hold is that $\Phi_{IP}(b, c_I)$ holds for some set I. Using the definition (21) of $\Phi_{IP}(b, c_I)$, it follows that $P(X,b) \Rightarrow \bigvee_{i \in I} Q_i(X_i, b_i, c_i)$ is true for all X and for $X = a$ in particular. But $P(a,b)$ holds. Therefore $Q_i(a_i, b_i, c_i)$ holds for some i.

To show necessity, assume that (23) holds and let b and c be values for Y and Z that make the premise in (22) true, that is, $R(b,c)$ hold. We need to show that either $(\forall X)\neg P(X,b)$ or $\Phi_{IP}(b, c_I)$ holds for some $I \subseteq \{1, \ldots, n\}$. So assume that $(\exists X)P(X,b)$ and let $V = \{X \mid P(X,b)\}$. Using (23), we can infer that for each $a' \in V$, there is some Q_i that "covers" the violation, *i.e.*, there is some i such that $Q_i(a'_i, b_i, c_i)$ holds. Now, consider the set I of all these i's that correspond to all a' in V. By construction of I, we know that $I \neq \emptyset$ (since $V \neq \emptyset$) and $(\forall X)(P(X,b) \Rightarrow \bigvee_{i \in I} Q_i(X_i, b_i, c_i))$ holds. This is exactly the second conjunct in $\Phi_{IP}(b, c_I)$. Furthermore, by construction of I again, we know that for all $i \in I$, there is a value $a' \in V$ that makes $Q_i(a'_i, b_i, c_i)$ true. In other words, the first conjunct in $\Phi_{IP}(b, c_I)$ also holds. Therefore, $\Phi_{IP}(b, c_I)$ holds. ∎

Theorem 4.3 *The following condition is a complete local test that the insertion of P(a,b) is safe:*

$$[\bigvee_{i \in L, Z_i = \emptyset} Q_i(a_i, b_i)] \vee [(\exists X)P(X,b) \wedge \bigwedge_{I \subseteq L, I \neq \emptyset} (\forall Z_I)[\Phi_{IP}(b, Z_I) \Rightarrow \bigvee_{i \in L, Z_i \subseteq Z_I} Q_i(a_i, b_i, Z_i)]]^4 \qquad (24)$$

∎

Proof: Again, the structure of the proof here is similar to the case of single negation.

To show soundness, assume there is no prior violation and assume the test (24) is satisfied, that is, either the first disjunct in (24) is satisfied or the second disjunct is satisfied. In either case, we need to show that $P(a,b)$ does not introduce any new violation. Let c be an arbitrary constant that satisfies $R(b,c)$. We need to show $Q_i(a_i, b_i, c_i)$ holds for some i.

In the first case, we are done since $Q_i(a_i, b_i)$ holds for some i such that $Z_i = \emptyset$. In the second case, $(\exists X)P(X,b)$ holds and by applying Lemma 4.2 for $Y = b$ and $Z = c$, we infer $\bigvee_{I \subseteq L, I \neq \emptyset} \Phi_{IP}(b, c_I)$. Using the implication in (24), we infer $\bigvee_{I \subseteq L, I \neq \emptyset}[\bigvee_{i \in L, Z_i \subseteq Z_I} Q_i(a_i, b_i, c_i)]$ which is logically equivalent to $\bigvee_{i \in L} Q_i(a_i, b_i, c_i)]$.

To show completeness, assume the test (24) is not satisfied. That is, $Q_i(a_i, b_i)$ is false for all i such that $Z_i = \emptyset$, and either $(\forall X)\neg P(X,b)$ or some implication in (24) is false. In either case, we need to show that we can choose an R that satisfies the constraint before the insertion but that violates it after.

In the first case, all we need is to pick a value c for Z whose components do not appear in any Q_i and choose $R = \{< b, c >\}$. Clearly no violation existed before the insertion since there was no tuple in P that joins with it. Also, inserting $P(a,b)$ causes a violation: for any i such that $Z_i \neq \emptyset$, c_i must have components not in Q_i (by construction of c) and thus $Q_i(a_i, b_i, c_i)$ is false; for any i such that $Z_i = \emptyset$, $Q_i(a_i, b_i)$ is false by hypothesis.

In the second case, there is a set I_0 and a value c_{I_0} for Z_{I_0} that falsify the implication: $\Phi_{I_0 P}(b, c_{I_0})$ holds but $Q_i(a_i, b_i, c_i)$ is false for all i such that $Z_i \subseteq Z_{I_0}$. Let us choose $R = \{< b, c >\}$ where c is constructed by extending c_{I_0} with component values not in any Q_i. Since $\Phi_{IP}(b, c_{I_0})$ holds, R satisfies condition (22) in Lemma 4.2. Thus, no violation existed before the insertion.

[4] Note that for a given I, the index i in the disjunction not only ranges over I, but may also take values k outside of I since there may be some $Z_k \subseteq Z_I$.

Furthermore, we claim that a violation is created by the insertion, that is, no $Q_i(a_i, b_i, c_i)$ can hold: for those i such that $Z_i \subseteq Z_{I_0}$, we already know that $Q_i(a_i, b_i, c_i)$ is false (by hypothesis); for those i such that $Z_i \not\subseteq Z_{I_0}$, c_i must have components that do not appear in Q_i (by construction of c) and thus $Q_i(a_i, b_i, c_i)$ is false again. ∎

Implementation

A Datalog program that implements the complete test (24) for safely inserting $P(a, b)$ is shown here (with $safeinsert$ as the query predicate):

$$
\begin{array}{rrcl}
(1) & local_I(Z_I) & \text{:-} & \bigwedge_{i\in I} P(X^{(i)}, b) \ \&\ Q_i(X_i^{(i)}, b_i, Z_i). \\
(2) & forbid_I(Z_I) & \text{:-} & local_I(Z_I) \ \&\ P(X, b) \ \&\ \bigwedge_{i\in I} \neg Q_i(X_i, b_i, Z_i). \\
(3) & quotient_I(Z_I) & \text{:-} & local_I(Z_I) \ \&\ \neg forbid_I(Z_I). \\
(4) & notcovered_I & \text{:-} & quotient_I(Z_I) \ \&\ \bigwedge_{i\in L \mid Z_i \subseteq Z_I} \neg Q_i(a_i, b_i, Z_i). \\
(5) & notcovered & \text{:-} & notcovered_I. \\
(6) & safeinsert & \text{:-} & P(X, b) \ \&\ \neg notcovered. \\
(7) & safeinsert & \text{:-} & Q_i(a_i, b_i).
\end{array}
$$

Rules (1) through (5) are actually templates that must be instantiated for every nonempty subset $I \subseteq L$. Given a particular I, the conjunctions in the body of (1) and (2) must be expanded over every value i in I, the conjunction in the body of (4) must be expanded over every value i in L such that $Z_i \subseteq Z_I$. Finally rule (7) is a template that must be instantiated for every value i in L such that $Z_i = \emptyset$.

Special case

This is the case where all Q_i's use the same Z-variables, say Z_1. This special case is mentioned here merely to show that the added complexity enters when we allow the Q_i's to use different Z variable sets. We only show the results, leaving the proof to the reader.

The complete local test in this special case reduces to

$$(\exists X)P(X, b) \wedge (\forall Z_1)(\Psi'_{QP}(b, Z_1) \Rightarrow \bigvee_{i\in L} Q_i(a_i, b_i, Z_1))$$

where Ψ'_{QP} here is defined to be

$$\Psi'_{QP}(Y, Z_1) \stackrel{\text{def}}{\equiv} (\exists X)[P(X, Y) \wedge \bigvee_{i\in L} Q_i(X_i, Y_i, Z_1)] \wedge (\forall X)[P(X, Y) \Rightarrow \bigvee_{i\in L} Q_i(X_i, Y_i, Z_1)]$$

We can see that this special case is a straightforward generalization of the case of single negation.

4.4 Negated subgoals with remote predicates

We now turn to cases where negated subgoals are allowed to use remote predicates as well. This section deals with the special case where all negated subgoals use remote predicates, that is, $L = \emptyset$. Consider constraints of the form

$$panic \text{ :- } \underbrace{P(X, Y)}_{local} \ \&\ \underbrace{\bigwedge_{j\in M} \neg Q_j(X_j, Y_j, Z_j) \ \&\ R(Y, Z)}_{remote}. \tag{25}$$

It turns out that in this case, unlike the case of negated subgoals with local predicates, the complete local test is a very simple one.

Theorem 4.4 *The following condition is a complete local test that the insertion of P(a,b) is safe:*

$$(\exists X)[P(X,b) \wedge \bigwedge_{j \in M} X_j = a_j] \tag{26}$$

∎

Proof: To prove soundness, assume that (26) is satisfied. In order to show that $P(a,b)$ does not introduce any violation, let c be a value for Z that makes $R(b,c)$ true. We need to show that $\bigvee_{j \in M} Q_j(a_j, b_j, c_j)$ holds. Let a' be a value for X that satisfies (26): $P(a', b)$ holds and $a'_j = a_j$ for all $j \in M$. Since there is no violation before, it must be the case that $\bigvee_{j \in M} Q_j(a'_j, b_j, c_j)$ holds. Since $a_j = a'_j$, this formula gives us what we need to prove.

To prove completeness, assume that (26) is not satisfied. We will construct relations Q_j's and R such that the given constraint is satisfied before the insertion of $P(a,b)$ but violated after. Let c be an arbitrary constant, $R = \{< b, c >\}$ and $Q_j = \{< a'_j, b_j, c_j > \ \mid \ < a', b > \in P \wedge a'_j \neq a_j\}$ for each $j \in M$. On the one hand, a violation is created by inserting $P(a,b)$ since by construction, if $Q_j(a'_j, b_j, c_j)$ holds, then necessarily $a'_j \neq a_j$. On the other hand, to show that no violation existed before, let a' be an arbitrary value for X such that $P(a', b)$ holds. Since (26) is not satisfied, then there must be an index $k \in M$ such that $a'_k \neq a_k$. By construction of the Q_j's, $< a'_k, b_k, c_k >$ must be in Q_k. In other words, $Q_k(a'_k, b_k, c_k)$ holds and no violation exists because of $P(a', b)$ for any a'. ∎

Implementation

We show a Datalog program that implements the test that the insertion of $P(a,b)$ is safe, where *safeinsert* is the query predicate:

$$safeinsert \quad :- \quad P(X,b) \ \& \ \textstyle\bigwedge_{j \in M} X_j = a_j.$$

4.5 Both local and remote negated subgoals present

This section extends all previous results by allowing some negated subgoals to use local predicates (i.e. $L \neq \emptyset$) and some negated subgoals to use remote predicates (i.e. $M \neq \emptyset$.) Consider constraints of the form

$$panic :- \underbrace{P(X,Y) \ \& \ \bigwedge_{i \in L} \neg Q_i(X_i, Y_i, Z_i)}_{local} \ \& \ \underbrace{\bigwedge_{j \in M} \neg Q_j(X_j, Y_j, Z_j) \ \& \ R(Y,Z)}_{remote}. \tag{27}$$

It turns out that in addition to negated subgoals that use local predicates, allowing negated subgoals to use remote predicates does not complicate the complete local test too much. The previous results can be easily generalized. We first state a Lemma that will be used to prove the Theorem for complete local test.

Lemma 4.3 *Let I be a nonempty subset of L, a and b be some constants* [5] *and let $\xi_{IPab}(Z_I)$ be defined as:*

$$(\exists X)(P(X,b) \wedge \bigwedge_{j \in M} X_j = a_j) \wedge \bigwedge_{i \in I} (\exists X_i) Q_i(X_i, b_i, Z_i) \wedge (\forall X)[(P(X,b) \wedge \bigwedge_{j \in M} X_j = a_j) \Rightarrow \bigvee_{i \in I} Q_i(X_i, b_i, Z_i)]$$

[5] Actually, a and b are vectors of constants.

If the constraint is not violated and if $(\exists X)[P(X,b) \wedge \bigwedge_{j \in M} X_j = a_j]$ holds, then the following condition necessarily holds:

$$(\forall Z)[R(b,Z) \wedge (\bigwedge_{j \in M} \neg Q_j(a_j, b_j, Z_j)) \Rightarrow \bigvee_{I \subseteq L, I \neq \emptyset} \xi_{IPab}(Z_I)]$$

∎

Proof:

Assume the constraint is satisfied and assume $(\exists X)[P(X,b) \wedge \bigwedge_{j \in M} X_j = a_j]$ holds. Let c be some arbitrary value for Z such that $R(b,c)$ holds and no $Q_j(a_j, b_j, c_j)$ holds for $j \in M$. We need to show $\xi_{IPab}(c_I)$ holds for some nonempty subset $I \subseteq L$. We will construct such a subset.

Let $V = \{X \mid P(X,b) \wedge \bigwedge_{j \in M} X_j = a_j\}$. By hypothesis $V \neq \emptyset$. For each $a' \in V$, there must be some $i \in L$ such that $Q_i(a'_i, b_i, c_i)$ holds, since otherwise $P(a',b)$ and $R(b,c)$ would be in violation. Consider the set I_0 of all these i's that correspond to some a' in V. By construction of I_0, $I_0 \neq \emptyset$ and $(\forall X)[X \in V \Rightarrow \bigvee_{i \in I_0} Q_i(X_i, b_i, c_i)]$ holds. This is exactly the third conjunct in $\xi_{I_0Pab}(c_{I_0})$. Also by construction of I_0, for any $i \in I_0$, there is a value $a' \in V$ that satisfies $Q_i(a'_i, b_i, c_i)$. In other words, the second conjunct in $\xi_{I_0Pab}(c_{I_0})$ also holds. Therefore $\xi_{I_0Pab}(c_{I_0})$ holds. ∎

Theorem 4.5 *The following condition is a complete local test that the insertion of P(a,b) is safe:*

$$[\bigvee_{i \in L, Z_i = \emptyset} Q_i(a_i, b_i)] \vee [(\exists X)[P(X,b) \wedge \bigwedge_{j \in M} X_j = a_j] \wedge \bigwedge_{I \subseteq L, I \neq \emptyset} (\forall Z_I)[\xi_{IPab}(Z_I) \Rightarrow \bigvee_{i \in L, Z_i \subseteq Z_I} Q_i(a_i, b_i, Z_i)]] \tag{28}$$

∎

Proof:

To show soundness, assume there is no violation before the insertion and that (28) holds. That is either the first disjunct in (28) is satisfied or the second disjunct is satisfied. Let c be some arbitrary constant that satisfies $R(b,c)$. In either case, we need to find some $Q_i(a_i, b_i, c_i)$ that holds.

In the first case, we are done since $Q_i(a_i, b_i)$ holds for some $i \in L$ such that $Z_i = \emptyset$. In the second case, we need to show either $\bigvee_{i \in L} Q_i(a_i, b_i, c_i)$ or $\bigvee_{j \in M} Q_j(a_j, b_j, c_j)$ holds. If the latter holds, we are done. Otherwise, we must show the former holds, assuming $\bigvee_{j \in M} Q_j(a_j, b_j, c_j)$ is false.

Applying Lemma 4.3, we infer that there is a nonempty subset $I_0 \subseteq L$ such that $\xi_{I_0Pab}(c_{I_0})$ holds. Using the implication in condition (28), we infer $\bigvee_{i \in L, Z_i \subseteq Z_{I_0}} Q_i(a_i, b_i, c_i)$ which is sufficient for what we need to prove.

To show completeness, assume (28) is not satisfied. That is, $Q_i(a_i, b_i)$ is false for all $i \in L$ such that $Z_i = \emptyset$, and either $(\exists X)[P(X,b) \wedge \bigwedge_{j \in M} X_j = a_j]$ is false or or it is true but some implication in (28) is false. In either case, we need to show that we can choose an R that satisfies the constraint before the insertion but that violates it after.

In the first case, let c be some value whose components do not appear in any local Q_i and let $R = \{< b, c >\}$ and $Q_j = \{< a'_j, b_j, c_j > \mid < a', b > \in P \wedge a'_j \neq a_j\}$ for every $j \in M$. On the one hand, inserting $P(a,b)$ guarantees a new violation: for any $j \in M$, $Q_j(a_j, b_j, c_j)$ is false (by construction of the remote Q_j's); for any $i \in L$ such that $Z_i \neq \emptyset$, c_i must have components not in Q_i (by construction of c) and thus $Q_i(a_i, b_i, c_i)$ is false; and for any $i \in L$ such that $Z_i = \emptyset$, $Q_i(a_i, b_i)$ is false by hypothesis. On the other hand, to show that no violation existed before, let a' be an arbitrary value for X such that $P(a',b)$ holds. By hypothesis, there must be an index $k \in M$ such that $a'_k \neq a_k$. By construction of the remote Q_j's, $Q_k(a'_k, b_k, c_k)$ must hold. In other words, no violation exists because of $P(a',b)$ for any a'.

In the second case, there is a set $I_0 \subseteq L$ and a value c_{I_0} for Z_{I_0} that makes $\xi_{I_0 Pab}(c_{I_0})$ true and $Q_i(a_i, b_i, c_i)$ false for all $i \in L$ such that $Z_i \subseteq Z_{I_0}$.

Let $R = \{< b, c >\}$, where c is constructed by extending c_{I_0} with values not in any local Q_i. For any $j \in M$, let $Q_j = \{< a'_j, b_j, c_j > \mid P(a', b) \wedge a'_j \neq a_j\}$

To show there is no prior violation, let a'' be some value for X such that $P(a'', b)$ holds. We need to find some i that makes $Q_i(a''_i, b_i, c_i)$ true. In the case where $\bigwedge_{j \in M} a''_j = a_j$, using the definition of $\xi_{I_0 Pab}(c_{I_0})$, we can infer $\bigvee_{i \in I_0} Q_i(a''_i, b_i, c_i)$. In the opposite case, there is some $k \in M$ such that $a''_k \neq a_k$, and by construction of the remote Q_j's, $Q_k(a''_k, b_k, c_k)$ must hold.

To show that a violation is created by the insertion, we need to show no $Q_i(a_i, b_i, c_i)$ can hold. Indeed, for any $j \in M$, $Q_j(a_j, b_j, c_j)$ is false by construction of the remote Q_j's. For any $i \in L$, there are two cases to consider here. If $Z_i \subseteq Z_{I_0}$, we know that $Q_i(a_i, b_i, c_i)$ is false since by hypothesis the union in (28) is false. If $Z_i \not\subseteq Z_{I_0}$, by construction of c, c_i must have components that do not appear in any local Q_i, and thus $Q_i(a_i, b_i, c_i)$ is false again. ∎

Implementation

A Datalog program that implements the complete test (28) for safely inserting $P(a, b)$ is shown here (with *safeinsert* as the query predicate):

(1)	$local'_I(Z_I)$	:-	$\bigwedge_{i \in I} P(X^{(i)}, b)$ & $[\bigwedge_{j \in M} X^{(i)}_j = a_j]$ & $Q_i(X^{(i)}_i, b_i, Z_i)$.
(2)	$forbid'_I(Z_I)$	:-	$local'_I(Z_I)$ & $P(X, b)$ & $[\bigwedge_{j \in M} X_j = a_j]$ & $\bigwedge_{i \in I} \neg Q_i(X_i, b_i, Z_i)$.
(3)	$quotient'_I(Z_I)$	:-	$local'_I(Z_I)$ & $\neg forbid'_I(Z_I)$.
(4)	$notcovered_I$	:-	$quotient'_I(Z_I)$ & $\bigwedge_{i \in L \mid Z_i \subseteq Z_I} \neg Q_i(a_i, b_i, Z_i)$.
(5)	$notcovered$	:-	$notcovered_I$.
(6)	$safeinsert$	:-	$P(X, b)$ & $[\bigwedge_{j \in M} X_j = a_j]$ & $\neg notcovered$.
(7)	$safeinsert$	:-	$Q_i(a_i, b_i)$.

Rules (1) through (5) are actually templates that must be instantiated for every nonempty subset $I \subseteq L$. Given a particular I, the conjunctions in the body of (1) and (2) must be expanded over every value i in I, the conjunction in the body of (4) must be expanded over every value i in L such that $Z_i \subseteq Z_I$. Finally rule (7) is a template that must be instantiated for every value i in L such that $Z_i = \emptyset$.

4.6 No positive remote subgoals

Finally, we consider the degenerate case where all subgoals with remote predicates are negated. Consider constraints of the form

$$panic :- \underbrace{P(X) \;\&\; \bigwedge_{i \in L} \neg Q_i(X_i)}_{local} \;\&\; \underbrace{\bigwedge_{j \in M} \neg Q_j(X_j)}_{remote}. \tag{29}$$

There is a simple form for our complete local test.

Theorem 4.6 *The following condition is a complete local test that the insertion of $P(a)$ is safe:*

$$(\exists X)[P(X) \wedge [\bigwedge_{j \in M} X_j = a_j] \wedge \bigwedge_{i \in L} \neg Q_i(X_i)] \vee \bigvee_{i \in L} Q_i(a_i) \tag{30}$$

∎

Proof: To show soundness, assume there is no prior violation and that test (30) is satisfied. We need to show that $P(a)$ does not cause any violation, that is, $Q_i(a_i)$ holds for some i. If $\bigvee_{i \in L} Q_i(a_i)$ holds, we are done. Otherwise, there must be some value a' for X that makes the first disjunct in (30) true. In other words, $P(a')$ and for all $j \in M$, $a'_j = a_j$ hold, and $Q_i(a'_i)$ is false for any $i \in L$. By hypothesis, $P(a')$ is not in violation. Thus $Q_k(a'_k)$ holds for some $k \in L \cup M$. Furthermore, k cannot be in L. Thus, $k \in M$ and $a'_k = a_k$ must hold. So $Q_k(a_k)$ holds.

To show completeness, assume test (30) is not satisfied. In other words, $Q_i(a_i)$ is false for all $i \in L$ and for any X in P, either $X_j \neq a_j$ for some $j \in M$ or $Q_i(X_i)$ holds for some $i \in L$. We need to choose a model for the remote Q_j's that satisfies the constraint before the insertion but that violates it after. For any $j \in M$, let $Q_j = \{< a'_j > \mid P(a') \wedge a'_j \neq a_j\}$. By construction of these remote Q_j's, no $Q_j(a_j)$ can hold for any $j \in M$. Also by hypothesis, $Q_i(a_i)$ is false for all $i \in L$. Therefore, inserting $P(a)$ causes the constraint to be violated. To show that there were no violation before the insertion, let a' be an arbitrary value for X such that $P(a')$ holds. By hypothesis, either $Q_i(a'_i)$ holds for some $i \in L$ or $a'_j \neq a_j$ for some $j \in M$. In the first case, we are done showing $P(a')$ did not cause any violation. In the second case, $Q_j(a'_j)$ must hold by construction of the remote Q_j's. So in this case again, $P(a')$ did not cause any violation. ∎

In the special case where there is no local negated subgoals (i.e. $L = \emptyset$,) the reader can verify that the complete local test can further be simplified into $(\exists X)[P(X) \wedge \bigwedge_{j \in M} X_j = a_j]$.

Implementation

A Datalog program that implements the complete test (30) for safely inserting $P(a)$ is shown here (with *safeinsert* as the query predicate):

(1) $safeinsert$:- $P(X, b)$ & $[\bigwedge_{j \in M} X_j = a_j]$ & $\bigwedge_{i \in L} \neg Q_i(X_i)$.
(2) $safeinsert$:- $Q_i(a_i)$.

Rule (2) is a template that must be instantiated for every value i in L.

5 Single deletions

In this section, we consider deletion of a single tuple from a single negated (local) subquery.

5.1 Single negated local subgoal

Recall the constraint given in (15) as

$$panic \text{ :- } \underbrace{P(X,Y) \ \& \ \neg Q(X, Y_1, Z)}_{local} \ \& \ \underbrace{R(Y,Z)}_{remote}.$$

Theorem 5.1 *The following condition is a CLT that the deletion of $Q(a, b_1, c)$ is safe:*

$$(\forall Y)[(P(a,Y) \wedge Y_1 = b_1) \Rightarrow \neg \Psi_{QP}(Y, c)] \tag{31}$$

∎

Proof: To show soundness, assume deletion of $Q(a, b_1, c)$ causes violation, i.e., there is a value b' for Y ($b'_1 = b_1$) that satisfies the constraint query, that is, $P(a, b')$ and $R(b', c)$ hold. Substituting b' for Y in (31), we infer $\neg \Psi_{QP}(b', c)$. Using the definition of Ψ_{QP}, we infer $(\exists X)[P(X, b') \wedge \neg Q(X, b'_1, c)]$ which, together with $R(b', c)$, implies a violation prior to the deletion.

To show completeness, assume condition (31) false. There is a value b' for Y that falsifies the implication. That is, $b'_1 = b_1$ and $P(a, b')$ both hold but $\Psi_{QP}(b', c)$ also holds. Let $R = \{< b', c >\}$. Using Lemma 4.1, we infer that no violation existed before. After the deletion of $Q(a, b_1, c)$, $P(a, b')$ and $R(b', c)$ clearly causes violation. ∎

Alternatively, the $\Psi_{QP}(Y, c)$ expression in the CLT can be replaced with $(\forall X)[P(X, Y) \Rightarrow Q(X, Y_1, c)]$.

Implementation

A Datalog program that implements the complete test (31) for safely deleting $Q(a, b_1, c)$ is shown here (with $safedelete$ as the query predicate):

$$\begin{array}{rcl}
local(Y) & :- & P(a, Y) \ \&\ Y_1 = b_1. \\
forbid(Y) & :- & local(Y) \ \&\ P(X, Y) \ \&\ \neg Q(X, b_1, c). \\
unsafe & :- & local(Y) \ \&\ \neg forbid(Y). \\
safedelete & :- & \neg unsafe.
\end{array}$$

5.2 Multiple negated local subgoals

Recall the constraint given in (20) as

$$panic :- \underbrace{P(X, Y) \ \&\ \bigwedge_{i \in L} \neg Q_i(X_i, Y_i, Z_i)}_{local} \ \&\ \underbrace{R(Y, Z)}_{remote}.$$

We consider deletion of $Q_h(a_h, b_h, c_h)$ [6] for some $h \in L$.

Let L^+ (resp. L^-) denotes the set of indices $i \in L$ such that $Z_i \subseteq Z_h$ (resp. $Z_i \not\subseteq Z_h$), and let us define the following:

$$\begin{aligned}
\phi^{(1)}_{a_h b_h c_h}(X, Y) &\stackrel{\text{def}}{\equiv} X_h = a_h \wedge Y_h = b_h \wedge \bigwedge_{i \in L^+, i \neq h} \neg Q_i(X_i, Y_i, c_{hi})^7 \\
\phi^{(2)}_{b_h c_h}(X, Y) &\stackrel{\text{def}}{\equiv} P(X, Y) \wedge \neg Q_h(X_h, b_h, c_h) \wedge \bigwedge_{i \in L^+, i \neq h} \neg Q_i(X_i, Y_i, c_{hi}) \\
\phi^{(3)}_{b_h c_h I}(Y, Z_I) &\stackrel{\text{def}}{\equiv} (\forall X')[\phi^{(2)}_{b_h c_h}(X', Y) \Rightarrow \bigvee_{i \in I} Q_i(X'_i, Y_i, Z_i)] \wedge [Z_{Ih} = c_{hI}]^8
\end{aligned}$$

Theorem 5.2 *The following condition is a CLT that the deletion of $Q_h(a_h, b_h, c_h)$ is safe:*

$$\begin{array}{rl}
(\forall XY) & [\,[P(X, Y) \wedge \phi^{(1)}_{a_h b_h c_h}(X, Y)] \Rightarrow (\exists X')\phi^{(2)}_{b_h c_h}(X', Y)] \wedge \\
\bigwedge_{I \subseteq L^-, I \neq \emptyset} (\forall XYZ_I) & [\,[P(X, Y) \wedge \phi^{(1)}_{a_h b_h c_h}(X, Y) \wedge \phi^{(3)}_{b_h c_h I}(Y, Z_I)] \Rightarrow \bigvee_{i \in L^-, Z_i \subseteq Z_I} Q_i(X_i, Y_i, Z_i)]
\end{array} \quad (32)$$

∎

[6] Note that a_h,b_h and c_h denote vectors of constants.

[7] Another notation to be clarified: c_{hi} denotes the projection of c_h onto the variables denotes by X_i. This projection makes sense since $X_i \subseteq X_h$.

[8] The notation $Z_{Ih} = c_{hI}$ means that the restriction of Z_I variables to Z_h be bound to the restriction of c_h to the Z_I variables. In case where Z_I and Z_h are mutually disjoint, the equality becomes vacuously true.

Proof: To show soundness, assume deletion of $Q(a_h, b_h, c_h)$ causes a violation of the constraint, i.e., there are extensions a, b, c of a_h, b_h, c_h respectively such that $P(a,b)$ and $R(b,c)$ hold but $Q_i(a_i, b_i, c_i)$ false for all $i \neq h$. So $\phi^{(1)}_{a_h b_h c_h}(a,b)$ holds.

By hypothesis, since the test condition is satisfied, if we substitute L^-,a,b,c_{L^-} for I,X,Y,Z_I respectively in (32), we infer

$$\phi^{(3)}_{b_h c_h L^-}(b, c_I) \Rightarrow \bigvee_{i \in L^-} Q_i(a_i, b_i, c_i)$$

Since $Q_i(a_i, b_i, c_i)$ is false for all $i \in L^-$, $\phi^{(3)}_{b_h c_h L^-}(b, c_I)$ cannot hold, i.e., there must be some X' such that $\phi^{(2)}_{b_h c_h}(X', b)$ holds but $Q_i(X'_i, b_i, c_i)$ is false for all $i \in L^-$. By expanding $\phi^{(2)}_{b_h c_h}(X', b)$, we infer $P(X', b)$ holds but $Q_i(X'_i, b_i, c_i)$ is false for all i in L^+. We just found a prior violation, which is contradictory.

To show completeness, assume the test is not satisfied, that is, either the first conjunct is false or one of the subsequent conjuncts is false. In either case, we need to show that we can choose an R that satisfies the constraint before the deletion but that violates it after.

In the first case, there are extensions a, b of a_h, b_h respectively that satisfy $P(a,b)$, $\phi^{(1)}_{a_h b_h c_h}(a,b)$ and $(\forall X')\neg\phi^{(2)}_{b_h c_h}(X', b)$. By expanding $\phi^{(1)}_{a_h b_h c_h}(a,b)$, we infer that $Q_i(a_i, b_i, c_i)$ is false for all i in L^+ but h. Let $R = \{< b, c >\}$ where c extends c_h with components not in any Q_i. After deletion of $Q_h(a_h, b_h, c_h)$, $P(a,b)$ and $R(b,c)$ are in violation since $Q_i(a_i, b_i, c_i)$ is false for any i in L^- (by construction of c) or in L^+ (as a consequence of $\phi^{(1)}_{a_h b_h c_h}(a,b)$). To verify there is no prior violation, assume $P(X', b)$ true for some X'. If $Q_i(X'_i, b_i, c_i)$ holds for some $i \in L^+, i \neq h$, we are done. Otherwise, since $\phi^{(2)}_{b_h c_h}(X', b)$ is false, $Q_h(X'_h, b_h, c_h)$ must hold.

In the second case, there are extensions a,b of a_h,b_h, some subset $I_0 \subseteq L^-$, and an instance d of Z_{I_0} consistent with c_h that satisfy $P(a,b)$, $\phi^{(1)}_{a_h b_h c_h}(a,b)$, $\phi^{(3)}_{b_h c_h I_0}(b, d)$ and $\bigwedge_{i \in L^-, Z_i \subseteq Z_{I_0}} \neg Q_i(a_i, b_i, d_i)$. Let $R = \{< b, c >\}$ where c extends c_h and d with components not in any Q_i. As an immediate consequence, $Q_i(a_i, b_i, c_i)$ is false for all i in L^- such that $Z_i \not\subseteq Z_{I_0}$. After deletion of $Q_h(a_h, b_h, c_h)$, $P(a,b)$ and $R(b,c)$ are in violation since $Q_i(a_i, b_i, c_i)$ is false for $i \in L^-, Z_i \not\subseteq Z_{I_0}$ (by construction of c), for $i \in L^-, Z_i \subseteq Z_{I_0}$ (by hypothesis) and for $i \in L^+, i \neq h$ (as a consequence of $\phi^{(1)}_{a_h b_h c_h}(a,b)$). To verify there is no prior violation, assume $P(X', b)$ true for some X'. If $Q_i(X'_i, b_i, c_i)$ holds for some $i \in L^+, i \neq h$, or if $Q_h(X'_h, b_h, c_h)$ holds, we are done. Otherwise, $\phi^{(2)}_{b_h c_h}(X', b)$ must hold and thus $\bigvee_{i \in I_0} Q_i(X'_i, b_i, d_i)$ holds (by expanding $\phi^{(3)}_{b_h c_h I_0}(b, d)$). Since d_i and c_i coincide over I_0, we are done. ∎

Implementation

A Datalog program that implements the complete test (32) for safely deleting $Q_h(a_h, b_h, c_h)$ is shown here (with $safedelete$ as the query predicate):

$$
\begin{array}{lrcl}
(1) & local(X,Y) & :- & P(X,Y)\ \&\ X_h = a_h\ \&\ Y_h = b_h\ \&\ \bigwedge_{i\in L^+, i\neq h} \neg Q_i(X_i, Y_i, c_{hi}). \\
(2) & phi^{(2)}(X,Y) & :- & P(X,Y)\ \&\ \neg Q_h(X_h, b_h, c_h)\ \&\ \bigwedge_{i\in L^+, i\neq h} \neg Q_i(X_i, Y_i, c_{hi}). \\
(3) & bounded(Y) & :- & phi^{(2)}(X,Y). \\
(4) & local_I(Y, Z_I) & :- & [\bigwedge_{i\in I} phi^{(2)}(X^{(i)}, Y)\ \&\ Q_i(X_i^{(i)}, Y_i, Z_i)]\ \&\ Z_{Ih} = c_{hI}. \\
(5) & forbid_I(Y, Z_I) & :- & local_I(Y, Z_I)\ \&\ phi^{(2)}(X,Y)\ \&\ \bigwedge_{i\in I} \neg Q_i(X_i, Y_i, Z_i). \\
(6) & phi_I^{(3)}(Y, Z_I) & :- & local_I(Y, Z_I)\ \&\ \neg forbid_I(Y, Z_I). \\
(7) & notcovered_I(X,Y) & :- & local(X,Y)\ \&\ phi_I^{(3)}(Y, Z_I)\ \&\ \bigwedge_{i\in L^-, Z_i\subseteq Z_I} \neg Q_i(X_i, Y_i, Z_i). \\
(8) & unsafe & :- & local(X,Y)\ \&\ \neg bounded(Y). \\
(9) & unsafe & :- & local(X,Y)\ \&\ notcovered_I(X,Y). \\
(10) & safedelete & :- & \neg unsafe.
\end{array}
$$

Rules (4)–(7) and (9) are actually templates that must be instantiated for every nonempty subset $I \subseteq L^-$.

5.3 Both negated local and remote subgoals present

Recall the constraint given in (27) as

$$
panic :-\ \underbrace{P(X,Y)\ \&\ \bigwedge_{i\in L} \neg Q_i(X_i, Y_i, Z_i)}_{local}\ \&\ \underbrace{\bigwedge_{j\in M} \neg Q_j(X_j, Y_j, Z_j)\ \&\ R(Y,Z)}_{remote}.
$$

Let L^+, L^-, $\phi^{(1)}_{a_h b_h c_h}$ and $\phi^{(2)}_{b_h c_h}$ be defined as in Theorem 5.2, and let us define the following:

$$
\phi^{(4)}_{b_h c_h I}(X, Y, Z_I) \ \stackrel{\text{def}}{\equiv}\ (\forall X')[(\phi^{(2)}_{b_h c_h}(X', Y) \wedge \bigwedge_{j\in M} X'_j = X_j) \Rightarrow \bigvee_{i\in I} Q_i(X'_i, Y_i, Z_i)] \wedge [Z_{Ih} = c_{hI}]
$$

Theorem 5.3 *The following condition is a CLT that the deletion of $Q_h(a_h, b_h, c_h)$ is safe:*

$$
(\forall XY)\quad [\,[P(X,Y) \wedge \phi^{(1)}_{a_h b_h c_h}(X,Y)] \Rightarrow (\exists X')(\phi^{(2)}_{b_h c_h}(X', Y) \wedge \bigwedge_{j\in M} X'_j = X_j)] \wedge
$$

$$
\bigwedge_{I\subseteq L^-, I\neq\emptyset} (\forall XYZ_I)\quad [\,[P(X,Y) \wedge \phi^{(1)}_{a_h b_h c_h}(X,Y) \wedge \phi^{(4)}_{b_h c_h I}(X, Y, Z_I)] \Rightarrow \bigvee_{i\in L^-, Z_i \subseteq Z_I} Q_i(X_i, Y_i, Z_i)] \quad (33)
$$

∎

Proof: The proof is very similar to the case where there are no negated remote subgoals. Among the needed adjustments, the most notable change involves choosing the state for the remote Q_j's when the test is not satisfied. For these remote Q_j's, choose $Q_j = \{< X_j, b_j, c_j > \mid P(X, b) \wedge X_j \neq a_j\}$. ∎

Implementation

A Datalog program that implements the complete test (33) for safely deleting $Q_h(a_h, b_h, c_h)$ is shown here (with *safedelete* as the query predicate):

$$\begin{array}{lrcl}
(1) & local(X,Y) & :- & P(X,Y) \mathbin{\&} X_h = a_h \mathbin{\&} Y_h = b_h \mathbin{\&} \bigwedge_{i\in L^+, i\neq h} \neg Q_i(X_i, Y_i, c_{hi}). \\
(2) & phi^{(2)}(X,Y) & :- & P(X,Y) \mathbin{\&} \neg Q_h(X_h, b_h, c_h) \mathbin{\&} \bigwedge_{i\in L^+, i\neq h} \neg Q_i(X_i, Y_i, c_{hi}). \\
(3) & bounded(X,Y) & :- & local(X,Y) \mathbin{\&} phi^{(2)}(X',Y) \mathbin{\&} \bigwedge_{j\in M} X'_j = X_j. \\
(4) & local_I(X,Y,Z_I) & :- & local(X,Y) \mathbin{\&} [\bigwedge_{i\in I} phi^{(2)}(X^{(i)},Y) \mathbin{\&} \bigwedge_{j\in M} X^{(i)}_j = X_j \mathbin{\&} Q_i(X^{(i)}_i, Y_i, Z_i)] \mathbin{\&} Z_{Ih} = c_{hI}. \\
(5) & forbid_I(Y,Z_I) & :- & local_I(X,Y,Z_I) \mathbin{\&} phi^{(2)}(X',Y) \mathbin{\&} \bigwedge_{j\in M} X'_j = X_j \mathbin{\&} \bigwedge_{i\in I} \neg Q_i(X'_i, Y_i, Z_i). \\
(6) & phi^{(4)}_I(X,Y,Z_I) & :- & local_I(X,Y,Z_I) \mathbin{\&} \neg forbid_I(X,Y,Z_I). \\
(7) & notcovered_I(X,Y) & :- & local(X,Y) \mathbin{\&} phi^{(4)}_I(X,Y,Z_I) \mathbin{\&} \bigwedge_{i\in L^-, Z_i \subseteq Z_I} \neg Q_i(X_i, Y_i, Z_i). \\
(8) & unsafe & :- & local(X,Y) \mathbin{\&} \neg bounded(X,Y). \\
(9) & unsafe & :- & local(X,Y) \mathbin{\&} notcovered_I(X,Y). \\
(10) & safedelete & :- & \neg unsafe.
\end{array}$$

Rules (4)–(7) and (9) are actually templates that must be instantiated for every nonempty subset $I \subseteq L^-$.

Interesting special cases

When a remote negated subgoal shares the same X variables as the local positive subgoal (that is, $\exists j \in M : X_j = X$), the CLT simplifies into:

$$\neg(\exists XY)(\phi_{c_h}(X,Y) \wedge X_h = a_h \wedge Y_h = b_h)$$

When, in addition to the above restrictions, every local negated subgoal (except the one under change) uses some Z variable that does not occur in the negated subgoal under change (i.e. $L^- = \emptyset$), the CLT further degenerates into:

$$\neg(\exists XY)(P(X,Y) \wedge X_h = a_h \wedge Y_h = b_h)$$

This is the most drastic requirement on the local relations in order to guarantee a safe deletion.

5.4 No positive remote subgoals

Recall the constraint given in (29) as

$$panic :- \underbrace{P(X) \mathbin{\&} \bigwedge_{i\in L} \neg Q_i(X_i)}_{local} \mathbin{\&} \underbrace{\bigwedge_{j\in M} \neg Q_j(X_j)}_{remote}.$$

Theorem 5.4 *The following condition is a CLT that the deletion of $Q_h(a_h)$ is safe:*

$$(\forall X)[(P(X) \wedge X_h = a_h \wedge \bigwedge_{i\in L, i\neq h} \neg Q_i(X_i)) \Rightarrow (\exists X')(P(X') \wedge \bigwedge_{j\in M} X'_j = X_j \wedge \bigwedge_{i\in L} \neg Q_i(X'_i))] \qquad (34)$$

∎

Proof: To show soundness, assume deletion of $Q_h(a_h)$ causes violation, i.e., there is an extension a of a_h such that $P(a)$ holds but $Q_i(a_i)$ is false for all $i \neq h$. By substituting a for X in condition (34), we infer there is an X' such that $P(X')$,no $Q_i(X'_i)$ holds for $i \in L$, and $X'_j = a_j$ for all $j \in M$. The latter implies that for all $j \in M$, $Q_j(X'_j)$ is false since $Q_j(a_j)$ is false. So X' is in violation, which contradicts the assumption of no prior violation.

To show completeness, assume the test is not satisfied. There is an extension a of a_h that satisfies $P(a)$, $\bigwedge_{i \in L, i \neq h} \neg Q_i(a_i)$ and

$$(\forall X')[(P(X') \wedge \bigwedge_{j \in M} X'_j = a_j) \Rightarrow \bigvee_{i \in L} Q_i(X'_i)]$$

For each $j \in M$, choose $Q_j = \{< X_j > \mid P(X) \wedge X_j \neq a_j\}$. After deletion of $Q_h(a_h)$, $P(a)$ is in violation since $Q_i(a_i)$ is false for all $i \in L$ (by hypothesis) and $Q_j(a_j)$ is false for all $j \in M$ (by construction of the remote Q_j's). To verify there is no prior violation, assume $P(X')$ true. If $\bigvee_{i \in L} Q_i(X'_i)$ holds, we are done. Otherwise, there must be some $k \in M$ such that $X'_k \neq a_k$. Therefore $Q_k(X'_k)$ holds (by construction of the remote Q_j's). ∎

Implementation

A Datalog program that implements the complete test (34) for safely deleting $Q_h(a_h)$ is shown here (with *safedelete* as the query predicate):

$$\begin{array}{rcl} local(X) & :- & P(X) \;\&\; X_h = a_h \;\&\; \bigwedge_{i \in L, i \neq h} \neg Q_i(X_i). \\ bounded(X) & :- & local(X) \;\&\; P(X') \;\&\; \bigwedge_{j \in M} X'_j = X_j \;\&\; \bigwedge_{i \in L} \neg Q_i(X'_i). \\ unsafe & :- & local(X) \;\&\; \neg bounded(X). \\ safedelete & :- & \neg unsafe. \end{array}$$

5.5 No positive local subgoals

To complete our results for deletion of tuples from negated subqueries, we need to consider an additional form of constraint as follows

$$panic :- \underbrace{\bigwedge_{i \in L} \neg Q_i(Z_i)}_{local} \;\&\; \underbrace{\bigwedge_{j \in M} \neg Q_j(Z_j) \;\&\; R(Z)}_{remote}.$$

Theorem 5.5 *Let $L^+ = \{i \mid i \in L \wedge Z_i \subseteq Z_h\}$. The following condition is a CLT that the deletion of $Q_h(c_h)$ is safe:*

$$\bigvee_{i \in L^+, i \neq h} Q_i(c_{hi}) \tag{35}$$

∎

Proof: To show soundness, assume deletion of $Q_h(c_h)$ causes violation, i.e., there is an extension c of c_h such that $R(c)$ holds but $Q_i(c_i)$ is false for all $i \neq h$. This contradicts the assumption that the test is satisfied.

To show completeness, assume the test is not satisfied, i.e., $Q_i(c_{hi})$ is false for all $i \in L^+, i \neq h$. Let $R = \{< c >\}$ where c extends c_h with component values not in any local Q_i. Let all remote Q_j's be empty. After deletion of $Q_h(c_h)$, $R(c)$ is in violation since $Q_i(c_i)$ is false for all $i \in L^+$ (by hypothesis), $Q_i(c_i)$ is false for all $i \in L, i \notin L^+$ (by construction of c) and $Q_j(c_j)$ is false all $j \in M$ (by construction of the remote Q_j's). To verify there is no prior violation, we just use the fact that $Q_h(c_h)$ holds prior to the deletion. ∎

6 Sets of Updates

In the previous section, we completely solved the problem of finding complete tests for single updates. In this section we solve the problem for certain sets of updates by either simple generalizations to the single update solutions or by simple decompositions in terms of the tests for single updates. For convenience, the local negated subgoals will be indexed by $L = \{1, \ldots, m\}$ and the remote negated subgoals by $M = \{m+1, \ldots, n\}$.

6.1 Sets of Insertions

Let ΔP be a set of tuples to be added to P and for each $i \in L$, let ΔQ_i be a set of tuples to be added to Q_i. Consider the problem of finding CLT's for the insertion update $(\Delta P, \Delta Q_1, \ldots, \Delta Q_m)$. Let Q_i^+ denote the relation Q_i after the update (that is, $Q_i^+ = Q_i \cup \Delta Q_i$).

Singleton ΔP's

Consider the special case where ΔP consists of a single tuple t. This case has already been treated previously but with the assumption that no Q_i changes. The extension to nonempty ΔQ_i turns out to be straightforward. Here, we simply state the results, leaving the proof to the reader (the previous proofs need to be slightly modified, especially when showing completeness).

- When the constraint query has both positive local and remote subgoals (i.e., both P and R nonvacuous,) a complete local test for safely adding $<a,b>$ to P and ΔQ_i to Q_i is very similar to (28), where some occurrences of Q_i are replaced with Q_i^+:

$$[\bigvee_{i \in L, Z_i = \emptyset} Q_i^+(a_i, b_i)] \vee (\exists X)[P(X, b) \wedge \bigwedge_{j \in M} X_j = a_j] \wedge \bigwedge_{I \subseteq L, I \neq \emptyset} (\forall Z_I)[\xi_{IPab}(Z_I) \Rightarrow \bigvee_{i \in L, Z_i \subseteq Z_I} Q_i^+(a_i, b_i, Z_i)]$$

- When the constraint query has no positive remote subgoal (i.e., vacuous R,) a complete local test for safely adding $<a>$ to P and ΔQ_i to local Q_i is very similar to (30):

$$(\exists X)[P(X) \wedge [\bigwedge_{j \in M} X_j = a_j] \wedge \bigwedge_{i \in L} \neg Q_i(X_i)] \vee \bigvee_{i \in L} Q_i^+(a_i)$$

Intuitively, the fact that the ΔQ_i's are not empty should not affect the bounds for the inaccessible states since these bounds are based on the accessible state prior to any update. The only change concerns the search for cover (for the new tuple to be added to P) that ought to take advantage of the larger Q_i^+'s.

Arbitrary ΔP's

The following theorem gives a decomposition of CLT for arbitrary ΔP's into CLT's for singleton ΔP's.

Theorem 6.1 *Insertion update (ΔP,ΔQ_1,...,ΔQ_m) is safe if and only if insertion update ($\{t\}$,ΔQ_1,...,ΔQ_m) is safe for all $t \in \Delta P$.* ∎

Proof:(Sketch) Consider the constraint query $E(P, Q_1, \ldots, Q_m)$ [9] for the constraint

$$panic :- P(X, Y) \ \& \ \bigwedge_{i \in L} \neg Q_i(X_i, Y_i, Z_i) \ \& \ \bigwedge_{j \in M} \neg Q_j(X_j, Y_j, Z_j) \ \& \ R(Y, Z).$$

[9] Technically, the constraint query is also a function of R and the remote Q_j's, but where there is no ambiguity, R and the Q_j's are dropped from our notation.

For any given P and local Q_i's, insertion update $(\Delta P, \Delta Q_1, \ldots, \Delta Q_m)$ is safe when

$$E(P, Q_1, \ldots, Q_m) = \emptyset \Rightarrow E(P \cup \Delta P, Q_1^+, \ldots, Q_m^+) = \emptyset \tag{36}$$

for all remote states. Also, using the particular form of E, it is easy to verify that

$$E(P \cup \Delta P, Q_1^+, \ldots, Q_m^+) = \bigcup_{t \in \Delta P} E(P \cup \{t\}, Q_1^+, \ldots, Q_m^+) \tag{37}$$

To show necessity, assume insertion update $(\Delta P, \Delta Q_1, \ldots, \Delta Q_m)$ is safe and let t' be an arbitrary tuple in ΔP. Consider a remote state such that $E(P, Q_1, \ldots, Q_m) = \emptyset$. We need to show $E(P \cup \{t'\}, Q_1^+, \ldots, Q_m^+) = \emptyset$. Using (36), we infer $E(P \cup \Delta P, Q_1^+, \ldots, Q_m^+) = \emptyset$. Using (37), we infer $E(P \cup \{t\}, Q_1^+, \ldots, Q_m^+) = \emptyset$ for all $t \in \Delta P$ and for t' in particular.

To show sufficiency, assume insertion $(\{t\}, \Delta Q_1, \ldots, \Delta Q_m)$ is safe for all $t \in \Delta P$. Consider a remote state such that $E(P, Q_1, \ldots, Q_m) = \emptyset$. We need to show $E(P \cup \Delta P, Q_1^+, \ldots, Q_m^+) = \emptyset$. It follows from the safety of $(\{t\}, \Delta Q_1, \ldots, \Delta Q_m)$ that $E(P \cup \{t\}, Q_1^+, \ldots, Q_m^+) = \emptyset$. Therefore, the left hand side of (37) is empty. ∎

6.2 Sets of Deletions

Let ΔP be a set of tuples to be removed from P and for each $i \in L$, let ΔQ_i be a set of tuples to be removed from Q_i. Consider the problem of finding CLT's for the deletion update $(\Delta P, \Delta Q_1, \ldots, \Delta Q_m)$. Let P^- denote the relation P after the update (that is, $P^- = P - \Delta P$).

Singleton ΔQ's

Consider the special case where a single tuple from only one of the local Q_i's is removed, say from Q_h. This case has already been treated previously but with the assumption that P does not change (i.e., ΔP empty). The extension to nonempty ΔP turns out to be straightforward. Here, we simply state the results, leaving the proof to the reader (the previous proofs need to be slightly modified, especially when showing completeness).

- When the constraint query has both positive local and remote subgoals (i.e., both P and R nonvacuous,) a complete test for safely removing ΔP from P and $\Delta Q_h = \{< a_h, b_h, c_h >\}$ from Q_h is very similar to (33), where some occurrences of P are replaced with P^-:

$$(\forall XY) \quad [\,[P^-(X,Y) \wedge \xi^{(1)}_{a_h b_h c_h}(X,Y)] \Rightarrow (\exists X')(\xi^{(2)}_{b_h c_h}(X',Y) \wedge \bigwedge_{j \in M} X'_j = X_j)] \wedge$$

$$\bigwedge_{I \subseteq L^-, I \neq \emptyset} (\forall XYZ_I) \quad [\,[P^-(X,Y) \wedge \xi^{(1)}_{a_h b_h c_h}(X,Y) \wedge \xi^{(3)}_{b_h c_h I}(X,Y,Z_I)] \Rightarrow \bigvee_{i \in L^-, Z_i \subseteq Z_I} Q_i(X_i, Y_i, Z_i)]$$

- When the constraint query has no positive remote subgoal (i.e., R vacuous,) a complete local test for safely removing ΔP from P and $\Delta Q_h = \{< a_h >\}$ from Q_h is very similar to (34):

$$(\forall X)[(P^-(X) \wedge X_h = a_h \wedge \bigwedge_{i \in L, i \neq h} \neg Q_i(X_i)) \Rightarrow (\exists X')(P(X') \wedge \bigwedge_{j \in M} X'_j = X_j \wedge \bigwedge_{i \in L} \neg Q_i(X'_i))]$$

Intuitively the fact that ΔP is not empty only affect the search for tuples in P that could potentially use the tuple to be deleted as cover. The search precisely takes advantage of a smaller P^-.

Deletion update $(\Delta P, \Delta Q_h)$

Theorem 6.2 *Deletion update $(\Delta P, \Delta Q_h)$ is safe if and only if deletion update $(\Delta P, \{t\})$ is safe for all $t \in \Delta Q_h$.* ▮

Proof:(Sketch) Deletion update $(\Delta P, \Delta Q_h)$ is safe when

$$E(P, Q_1, \ldots, Q_m) = \emptyset \Rightarrow E(P^-, Q_1, \ldots, Q_{h-1}, Q_h - \Delta Q_h, Q_{h+1}, \ldots, Q_m) = \emptyset \tag{38}$$

for all remote states. Also, using the particular form of E, it is easy to verify

$$E(P^-, Q_1, \ldots, Q_{h-1}, Q_h - \Delta Q_h, Q_{h+1}, \ldots, Q_m) = \bigcup_{t \in \Delta Q_h} E(P^-, Q_1, \ldots, Q_{h-1}, Q_h - \{t\}, Q_{h+1}, \ldots, Q_m) \tag{39}$$

The proof follows from the fact that the left hand side of (39) is empty if and only if every union member in the right hand side of (39) is empty. ▮

Deletion update $(\Delta Q_1, \ldots, \Delta Q_m)$

So far, when we say an update is safe, we implicitly mean its safety be evaluated in state $(P, Q_1, \ldots, Q_n, R)$ (hereafter referred to as S_0). In the subsequent, we allow safety to be evaluated in a state S that is not necessarily S_0 and to avoid ambiguity, we will explicitly mention the state in which an update's safety is to be evaluated. For instance, deletion update (ΔQ_2) is said to be safe in the state that results from applying deletion update (ΔQ_1) to S_0 when the following holds:

$$E(P, Q_1^-, Q_2, \ldots, Q_m) = \emptyset \Rightarrow E(P, Q_1^-, Q_2^-, Q_3, \ldots, Q_m) = \emptyset$$

Theorem 6.3 *Deletion update $(\Delta Q_1, \ldots, \Delta Q_m)$ is safe in S_0 if and only if for $i = 1, \ldots, m$, deletion update (ΔQ_i) is safe in the state that results from applying deletion update $(\Delta Q_1, \ldots, \Delta Q_{i-1})$ to S_0.* ▮

Proof:(Sketch) To show the theorem, we use the following graphs to illustrate the essence of the proof:

$$\begin{tikzcd}
S_0 \arrow[d, "H"'] & \cdots \arrow[l, "\subseteq"'] & S_{i-1} \arrow[l, "\subseteq"'] \arrow[d, "?"] \\
S_m \arrow[r, "\subseteq"] & \cdots \arrow[r, "\subseteq"] & S_i
\end{tikzcd}
\qquad\qquad
\begin{tikzcd}
S_0 \arrow[r, "H"] \arrow[d, "?"'] & \cdots \arrow[r, "H"] & S_{i-1} \arrow[d, "H"] \\
S_m & \cdots \arrow[l, "H"'] & S_i \arrow[l, "H"']
\end{tikzcd}$$

In these graphs, S_i denotes the result of applying deletion update $(\Delta Q_1, \ldots, \Delta Q_i)$ to state S_0, that is, state $(P, Q_1^-, \ldots, Q_i^-, Q_{i+1}, \ldots, Q_n, R)$. There is a directed edge from a state to another when consistency of the former implies consistency of the latter. We will refer this implication as *Consistency Implication.* An edge labeled with "$\subseteq$" indicates that the implication always holds because the set of tuples that violate the constraint in one state always contains the set in the other state. For example, because E is anti-monotonic in the Q_i's, $E(P, Q_1, \ldots, Q_m) \subseteq E(P, Q_1^-, Q_2, \ldots, Q_m)$ holds, thus, there is an edge from S_1 to S_0. An edge labeled with "H" denotes an implication true by hypothesis. An edge labeled with "?" denotes an implication to prove.

Since consistency implication is transitive, to show that consistency of one state implies that of another state, it is sufficient to find a directed path going from the former state to the latter.

The graph on the left shows necessity, that is, for any $i = 1, \ldots, m$, there is always a directed path from S_{i-1} to S_i using edges labeled with either "H" or "$\subseteq$". Similarly, the graph on the right shows sufficiency, that is, there is a directed path from S_0 to S_m. ∎

Deletion update $(\Delta P, \Delta Q_1, \ldots, \Delta Q_m)$

When the update consists of deletions from two or more Q_i's as well as from P, there does not seem to be an easy way to express its safety in terms of safety of simpler deletion updates for which we have a CLT.

EXAMPLE 6.1 To illustrate this difficulty, we consider the problem of expressing safety of deletion update $(\Delta P, \Delta Q_1, \Delta Q_2)$ as a conjunction of safety of deletion updates that do not have more than one ΔQ_i's. As before, nodes in the following graph

$$\begin{tikzcd}
 & (P^-, Q_1, Q_2) & \\
(\boldsymbol{P}, \boldsymbol{Q}_1, \boldsymbol{Q}_2) = \boldsymbol{S}_0 \arrow[ur] & (P^-, Q_1^-, Q_2) \arrow[u] & (P^-, Q_1, Q_2^-) \arrow[ul] \\
(P, Q_1^-, Q_2) \arrow[u] \arrow[ur] & (P, Q_1, Q_2^-) \arrow[ul] \arrow[ur] & (\boldsymbol{P}^-, \boldsymbol{Q}_1^-, \boldsymbol{Q}_2^-) = \boldsymbol{S}_2^- \arrow[ul] \arrow[u] \\
 & (P, Q_1^-, Q_2^-) \arrow[ul] \arrow[u] \arrow[ur] &
\end{tikzcd}$$

denote states that result from applying various subsets of deletion update $(\Delta P, \Delta Q_1, \Delta Q_2)$ to state S_0. Edges in the graph denote valid consistency implications due to containment of constraint queries. When the graph is viewed as a cube, the problem is to find a path connecting the two vertices in boldface. The path shall consist of edges chosen among the ones already shown (implicitly labeled with "$\subseteq$") and among other edges (not shown, aka H-edges) which correspond to safety of deletion updates that do not have more than one ΔQ_i's and which are the subject of discussion here.

On the one hand, in order to assure necessity, we must be able to prove each conjunct from the only hypothesis that $(\Delta P, \Delta Q_1, \Delta Q_2)$ is safe. In other words, the H-edges necessarily start from the plane that contains P's and end in the opposite plane (containing only P^-'s).

On the other hand, in order to assure sufficiency, we must find a path that starts from S_0 and ends at S_2^-. Now the problem is with choosing the first edge on the path: there are only three possible edges and each of theses edges ends in the plane containing only P^-'s at a vertex other than S_2^- from which there is no way to reach S_2^-. ∎

7 Conclusion

In this paper, we consider CQC$^\neg$ for which, while an important class of constraints, the problem of finding CLT remains unsolved. We solved the CLT problem for general CQC$^\neg$ constraints for single updates when inaccessible predicates have single occurrences in the constraint queries. We showed the various CLT solutions in a form that can easily be implemented in traditional query languages. 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 accessible relations. For general deletion updates and mixed updates, the example involving deletion update $(\Delta P, \Delta Q_1, \Delta Q_2)$ suggests that complete tests need to be developed for simultaneous single tuple updates to several subqueries.

Throughout this paper, we have assumed single occurrence of the inaccessible predicates in the constraint query. This restriction was needed to ensure that it is always possible to choose a remote state that can achieve specific instances for the subqueries R and Q_j's. If the restriction is removed, while all tests remain sound, their completeness can no longer be guaranteed. Indeed, if we allow multiple occurrences of the inaccessible 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 (see Appendix for more details).

The results achieved in this paper are not limited to conjunctive-query constraints. In fact, P and the Q_i's may be any query not involving inaccessible EDB's, as long as their delta sets can be computed, and R and the Q_j's may be any query not involving accessible EDB's, as long as their instances can be arbitrarily realized. Also, if a subquery involves both types of EDB's, the latter can often be separated by expanding the subquery in the constraint query.

References

[Gupta94] Gupta A.: *Partial Information Based Integrity Constraint Checking.* PhD Thesis, Stanford University, November 1994.

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

[GW93] Gupta A., Widom J.: Local Verification of Global Integrity Constraints in Distributed Databases. *Proceedings of the ACM SIGMOD International Conference on Management of Data,* 1993, pp. 49–58.

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

[Tiwari93] Tiwari S., Howard H. C.: Constraint Management on Distributed AEC Databases. *Fifth International Conf. on Computing in Civil and Building Engineering,* ASCE, 1993, pp. 1147–1154.

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

A Appendix

A.1 Determining query changes from EDB's updates

Let $p_1, \ldots, p_m$ be the predicates for EDB relations $P_1, \ldots, P_m$. Consider a conjunctive query

$$Q : q(X) :- \bigwedge_{i=1}^{n} G_i$$

where G_i denotes subgoal $p_{j_i}(X_i)$, with $1 \leq j_i \leq m$ and X_i being a vector whose components are either constants or variables. Let ΔP_j^+ (resp. ΔP_j^-) be the set of tuples to be inserted into (resp. deleted from) relation P_j, with predicate δp_j^+ (resp. δp_j^-). Assume that $\Delta P_j^+ \cap \Delta P_j^- = \emptyset$ (otherwise, there is some ambiguity as to whether some tuple is to be inserted or deleted). Also assume that $\Delta P_j^- \subseteq P_j$ (i.e. one cannot delete a tuple that is not there). Given the ΔP_j^+'s and ΔP_j^-'s, we want to determine ΔQ^+ and ΔQ^-, the net addition to and deletion from the answers to query Q (again assuming $\Delta Q^+ \cap \Delta Q^- = \emptyset$).

Insertions only

Let S be a nonempty set of subgoals G_i's. Define Q_S to be the query obtained from Q by replacing any subgoal $G_i \in S$ with $\delta p_{j_i}^+(X_i)$. The following is an expression that determines $\delta q^+(X)$:

$$[\bigvee_S q_S(X)] \wedge \neg q(X)$$

where S ranges over all nonempty sets of subgoals G_i's.

Deletions only

Let Q_i denote the query obtained from Q by replacing the subgoal G_i with $\delta p_{j_i}^-(X_i)$. Let Q^- denote the query obtained from Q by replacing any subgoal G_i with $p_{j_i}^(X_i) \wedge \neg\delta p_{j_i}^-(X_i)$.

The following is an expression that determines $\delta q^-(X)$:

$$[\bigvee_{i=1}^n q_i(X)] \wedge \neg q^-(X)$$

Both Insertions and Deletions

There are two ways to determine the net changes to Q, either by first "performing" the insertions followed by the deletions or the other way around. Since the order is irrelevant, both methods should yield identical results.

Let us define $\delta q'^-(X)$ as

$$[\bigvee_{i=1}^n q_i(X)] \wedge \neg q^-(X)$$

and $\delta q'^+(X)$ as

$$[\bigvee_S q'_S(X)] \wedge \neg q^-(X)$$

where Q'_S is obtained from Q by replacing any subgoal $G_i \in S$ with subgoal $\delta p_{j_i}^+(X_i)$, and any subgoal $G_i \notin S$ with $p_{j_i}(X_i) \wedge \neg\delta p_{j_i}^-(X_i)$.

Then $\delta q^+(X)$ is determined by $\delta q'^+(X) \wedge \neg\delta q'^-(X)$ and $\delta q^-(X)$ is determined by $\delta q'^-(X) \wedge \neg\delta q'^+(X)$.

Special case

When all subgoal's variables are distinguished (i.e. appear in $q(X)$) and $\Delta P_i \cap P_i = \emptyset$, one can show that testing potential changes for membership in the "final query answer" can be dropped without affecting the results. In other words:

- In the case with insertions only, $\delta q^+(X)$ can be simplified to $\bigvee_S q_S(X)$.
- In the case with deletions only, $\delta q^-(X)$ can be simplified to $\bigvee_{i=1}^n q_i(X)$.
- In the case with both insertions and deletions, similar simplifications can be applied to $\delta q'^+(X)$ and $\delta q'^-(X)$.

EXAMPLE A.1 Consider the query $q(X,Y)$:- $r(X,Y)$ & $r(Y,Z)$ & $s(X,Z)$. Given Δ^+R,Δ^-R,Δ^+S,Δ^-S, the following Datalog program computes the net changes to Q:

$$
\begin{array}{rcl}
q^-(X,Y) & :- & r(X,Y) \ \& \ r(Y,Z) \ \& \ s(X,Z) \ \& \ \neg\delta r^-(X,Y) \ \& \ \neg\delta r^-(Y,Z) \ \& \ \neg\delta s^-(X,Z). \\
potq^+(X,Y) & :- & \delta r^+(X,Y) \ \& \ r(Y,Z) \ \& \ s(X,Z) \ \& \ \neg\delta r^-(Y,Z) \ \& \ \neg\delta s^-(X,Z). \\
potq^+(X,Y) & :- & r(X,Y) \ \& \ \delta r^+(Y,Z) \ \& \ s(X,Z) \ \& \ \neg\delta r^-(X,Y) \ \& \ \neg\delta s^-(X,Z). \\
potq^+(X,Y) & :- & r(X,Y) \ \& \ r(Y,Z) \ \& \ \delta s^+(X,Z) \ \& \ \neg\delta r^-(X,Y) \ \& \ \neg\delta r^-(Y,Z). \\
potq^+(X,Y) & :- & \delta r^+(X,Y) \ \& \ \delta r^+(Y,Z) \ \& \ s(X,Z) \ \& \ \neg\delta s^-(X,Z). \\
potq^+(X,Y) & :- & \delta r^+(X,Y) \ \& \ r(Y,Z) \ \& \ \delta s^+(X,Z) \ \& \ \neg\delta r^-(Y,Z). \\
potq^+(X,Y) & :- & r(X,Y) \ \& \ \delta r^+(Y,Z) \ \& \ \delta s^+(X,Z) \ \& \ \neg\delta r^-(X,Y). \\
potq^+(X,Y) & :- & \delta r^+(X,Y) \ \& \ \delta r^+(Y,Z) \ \& \ \delta s^+(X,Z). \\
potq^-(X,Y) & :- & \delta r^-(X,Y) \ \& \ r(Y,Z) \ \& \ s(X,Z). \\
potq^-(X,Y) & :- & r(X,Y) \ \& \ \delta r^-(Y,Z) \ \& \ s(X,Z). \\
potq^-(X,Y) & :- & r(X,Y) \ \& \ r(Y,Z) \ \& \ \delta s^-(X,Z). \\
\delta q'^+(X,Y) & :- & potq^+(X,Y) \ \& \ \neg q^-(X,Y). \\
\delta q'^-(X,Y) & :- & potq^-(X,Y) \ \& \ \neg q^-(X,Y). \\
\delta q^+(X,Y) & :- & \delta q'^+(X,Y) \ \& \ \neg\delta q'^-(X,Y). \\
\delta q^-(X,Y) & :- & \delta q'^-(X,Y) \ \& \ \neg\delta q'^+(X,Y).
\end{array}
$$

If the query is changed to $q(X,Y,Z)$:- $r(X,Y)$ & $r(Y,Z)$ & $s(X,Z)$, and assuming that all insertions into R and S are effective, the program simplifies to:

$$
\begin{array}{rcl}
potq^+(X,Y,Z) & :- & \delta r^+(X,Y) \ \& \ r(Y,Z) \ \& \ s(X,Z) \ \& \ \neg\delta r^-(Y,Z) \ \& \ \neg\delta s^-(X,Z). \\
potq^+(X,Y,Z) & :- & r(X,Y) \ \& \ \delta r^+(Y,Z) \ \& \ s(X,Z) \ \& \ \neg\delta r^-(X,Y) \ \& \ \neg\delta s^-(X,Z). \\
potq^+(X,Y,Z) & :- & r(X,Y) \ \& \ r(Y,Z) \ \& \ \delta s^+(X,Z) \ \& \ \neg\delta r^-(X,Y) \ \& \ \neg\delta r^-(Y,Z). \\
potq^+(X,Y,Z) & :- & \delta r^+(X,Y) \ \& \ \delta r^+(Y,Z) \ \& \ s(X,Z) \ \& \ \neg\delta s^-(X,Z). \\
potq^+(X,Y,Z) & :- & \delta r^+(X,Y) \ \& \ r(Y,Z) \ \& \ \delta s^+(X,Z) \ \& \ \neg\delta r^-(Y,Z). \\
potq^+(X,Y,Z) & :- & r(X,Y) \ \& \ \delta r^+(Y,Z) \ \& \ \delta s^+(X,Z) \ \& \ \neg\delta r^-(X,Y). \\
potq^+(X,Y,Z) & :- & \delta r^+(X,Y) \ \& \ \delta r^+(Y,Z) \ \& \ \delta s^+(X,Z). \\
potq^-(X,Y,Z) & :- & \delta r^-(X,Y) \ \& \ r(Y,Z) \ \& \ s(X,Z). \\
potq^-(X,Y,Z) & :- & r(X,Y) \ \& \ \delta r^-(Y,Z) \ \& \ s(X,Z). \\
potq^-(X,Y,Z) & :- & r(X,Y) \ \& \ r(Y,Z) \ \& \ \delta s^-(X,Z). \\
\delta q^+(X,Y,Z) & :- & potq^+(X,Y) \ \& \ \neg potq^-(X,Y). \\
\delta q^-(X,Y,Z) & :- & potq^-(X,Y) \ \& \ \neg potq^+(X,Y).
\end{array}
$$

$\blacksquare$

A.2 Multiple Occurrences of Remote Predicates

When we allow remote predicates to have multiple occurrences in the constraint query, all CLT's we derived previously are still sound but are no longer necessarily complete. Referring to the proofs of completeness, the assumption of single occurrence of remote predicates was used precisely to guarantee that we can always "choose" a "counterexample" for the remote relations such that the remote query answer set includes some arbitrarily given set of tuples and excludes others.

EXAMPLE A.2 Consider the constraint $panic$:- $p(X, Y)$ & $r(X, Y)$ & $r(Y, X)$ where p is the predicate of a local relation and r that of a remote one. Let $R(X, Y)$ denote the remote query $r(X, Y) \wedge r(Y, X)$. Consider the insertion of $p(a, b)$ and the test that would have been complete if there were no multiple occurrences of remote predicates within R, that is, the condition "is $< a, b >$ already in p?". This condition, while still sufficient to guarantee safety of the insertion, is no longer necessary. In fact, there is another condition, namely that $< b, a >$ is in p, that can also guarantee safety of the insertion of $p(a, b)$.

The problem is that when looking for a counterexample, we can no longer assume that we can always find instances of relation r such that the remote query R contains *exactly* the tuple $<a,b>$. In this example, the minimal realizable R would contain both $<a,b>$ and $<b,a>$. $\blacksquare$

Example A.2 suggests that multiple occurrences of remote predicates (e.g. r) in the constraint query induce some kind of dependencies in the relation for the remote query (e.g. R) that must be captured in the condition of a complete test.

Definition A.1 (Closure of a relation under Conjunctive Query): Let $Q : q(Y)$:- $R(Y, Z)$ be a conjunctive query and let K be a set of constant vectors of same arity as predicate q. The *closure* of K under query Q, denoted $\mathcal{C}_Q(K)$, is a subset of the answers to query Q applied to a database D defined as follows:

- Assume $K = \{y_1, \ldots, y_n\}$. For each $i = 1, \ldots, n$, let z_i be a constant vector of same arity as Z and whose components are all distinct from each other and are all brand new; $R(y_i, z_i)$ is a conjunction of ground atoms; let B_i denote the set of these atoms. The database D is $\bigcup_{i=1}^{n} B_i$.
- $\mathcal{C}_Q(K)$ consists of those answers that do not contain component constants from any z_i's.

$\blacksquare$

Intuitively, $\mathcal{C}_Q(K)$ consists of all those tuples that are always in the answer set of query Q independently of the underlying EDB's, provided that the answer set contains K.

EXAMPLE A.3 Consider the remote query in Example A.2, $Q : q(X, Y)$:- $r(X, Y)$ & $r(Y, X)$. In the definition of the closure of $\{ab, ac\}$ under Q, database D consists of the atoms $r(a, b)$, $r(b, a)$, $r(a, c)$, $r(c, a)$. Applying Q to this database yields $\{ab, ba, ac, ca\}$ which defines the closure. Consider a similar query but where Y is projected out, $Q' : q(X)$:- $r(X, Y)$ & $r(Y, X)$. Suppose we want to find the closure of $\{a\}$ under Q'. With b as a brand new constant, database $D = \{ab, ba\}$ and applying Q' to D yields $\{a, b\}$. After filtering out answer b which is a new constant, we obtain $\mathcal{C}_{Q'}(\{a\}) = \{a\}$.
$\blacksquare$

We now state some formal properties of closure in the following lemma.

Lemma A.1 *Let $Q : q(Y)$:- $R(Y,Z)$ be a conjunctive query and let b be a constant vector of same arity as q. Let q_b^* denote the predicate for $\mathcal{C}_Q(\{b\})$. The following conditions always hold:*

- *$q_b^*(b)$, that is, in more general terms, the closure of any set of constants always contains the set itself.*
- *$(\forall YZ)[(q_b^*(Y) \wedge R(b,Z)) \Rightarrow (\exists Z')R(Y,Z')]$, that is, if the projection of R on Y contains b, then it must also contain the entire closure.*
- *Given a set K of constants of same arity as q, we can always choose a database instance for the underlying EDB's such that applying query Q on this database produces exactly $\mathcal{C}_Q(K)$ as answers.*

∎

CLT for the case of no negated subgoals

Recall the constraint defined in (13) as:

$$panic :- \underbrace{P(X,Y)}_{local} \ \& \ \underbrace{R(Y,Z)}_{remote}.$$

Theorem A.1 *Let q_b^* be the predicate for the closure of $\{b\}$ under query $Q : q(Y)$:- $R(Y,Z)$. The following is a CLT for safe insertion of $P(a,b)$:*

$$(\exists XY)[P(X,Y) \wedge q_b^*(Y)] \tag{40}$$

∎

Proof: To show soundness, assume (40) is satisfied, that is, there are constants a' and b' (not necessarily distinct from a and b) such that both $P(a',b')$ and $q_b^*(b')$ hold. Suppose that inserting $P(a,b)$ causes a violation, that is, there is some c such that $R(b,c)$ holds. Since $q_b^*(b')$ holds, according to Lemma A.1, there must be some c' such that $R(b',c')$ holds, which contradicts the assumption of no prior violation.

To show completeness, assume (40) is false, that is, $(\forall Y)[q_b^*(Y) \Rightarrow \neg(\exists X)P(X,Y)]$. Let c be a vector of distinct constants that are all brand new. Let us choose R to be the closure of $\{< b,c >\}$ under query $R(Y,Z)$. After insertion, $P(a,b)$ and $R(b,c)$ are clearly in violation. Before insertion, for each tuple $< b',c' >$ in R, we want to show there is no matching tuple in P. If b' has some c components, it is obvious there is no matching tuple in P. Otherwise, $q_b^*(b')$ must hold. Thus, using the hypothesis, we infer $(\exists X)P(X,b')$ which means there is no matching tuple in P. ∎

CLT for the case of no local negated subgoals

Recall the constraint given in (25) as:

$$panic :- \underbrace{P(X,Y)}_{local} \ \& \ \underbrace{\bigwedge_{j\in M} \neg Q_j(X_j,Y_j,Z_j) \ \& \ R(Y,Z)}_{remote}.$$

Let $X' \approx_X X''$ denote the fact X' and X'' agree over all X_j, that is, $\bigwedge_{j\in M} X'_j = X''_j$ ($\approx_Y$ and $\approx_Z$ denote similar notions).

Definition A.2 (Strict Closure): Let $Q : q(Y) :- R(Y, Z)$ be a conjunctive query and let b be a vector of constants of same arity as q. The *strict closure* of b under query Q denoted $\mathcal{C}_Q^+(K)$, is a subset of the answers to query Q applied to a database defined as follows:

- Let c be a constant vectors of same arity as Z and whose components are all distinct from each other and are all brand new. $R(b, c)$ is a conjunction of ground atoms and the database in question is this set of atoms.

- $\mathcal{C}_Q^+(K)$ consists of those answers y that not only do not contain any z's component constants (as in closure) but also satisfy two additional properties: $y \approx_Y b$ and there is some z such that $z \approx_Z c$ and $< y, z >$ is in R.

▮

Theorem A.2 *Let q_b^+ be the predicate for the strict closure of b under query $Q : q(Y) :- R(Y, Z)$. If no two Q_i's use the same predicate, and predicates used in the Q_i's do not occur in R, then the following is a CLT for safe insertion of $P(a, b)$:*

$$(\exists XY)[P(X, Y) \wedge q_b^+(Y) \wedge X \approx_X a] \tag{41}$$

▮

Proof: To prove soundness, assume that (41) is satisfied, that is, there are constants a' and b' (not necessarily distinct from a and b) such that $P(a', b')$, $q_b^+(b')$ and $a' \approx_X a$ hold. Suppose that inserting $P(a, b)$ causes a violation, that is, there is some constant c such that $R(b, c)$ holds but $Q_j(a_j, b_j, c_j)$ is false for all j. Since $q_b^+(b')$ holds, $b' \approx_X b$ holds and $\exists c'$ such that $R(b', c')$ and $c' \approx_Z c$ hold. Therefore $Q_j(a'_j, b'_j, c'_j)$ is false for all j, which contradicts the assumption of no prior violation.

To prove completeness, assume that (41) is false. Let c be a vector of distinct constants that are all brand new. Let us choose R to be the closure of $\{< b, c >\}$ under query $R(Y, Z)$. R can be partitioned into four sets R_1, R_2, R_3 and R_4 defined as follows:

$$\begin{aligned}
R_1 &= \{< b', c' > \mid b' \textit{ has no } c - \textit{component}, b' \approx_Y b, (\exists c'')\ [c'' \approx_Z c \wedge R(b', c'')]\} \\
R_2 &= \{< b', d' > \mid b' \textit{ has no } c - \textit{component}, b' \approx_Y b, (\forall c'')\ [R(b', c'') \Rightarrow c'' \not\approx_Z c]\} \\
R_3 &= \{< b'', d'' > \mid b'' \textit{ has no } c - \textit{component}, b'' \not\approx_Y b\} \\
R_4 &= \{< b''', d''' > \mid b''' \textit{ has some } c - \textit{component}\}
\end{aligned}$$

For each $j \in M$, let us choose Q_j to be the union of $S_j^{(1)}$, $S_j^{(2)}$ and $S_j^{(3)}$ defined as follows:

$$\begin{aligned}
S_j^{(1)} &= \{< a'_j, b'_j, c'_j > \ \mid P(a', b') \wedge R_1(b', c') \wedge a'_j \neq a_j\} \\
S_j^{(2)} &= \{< a'_j, b'_j, d'_j > \ \mid P(a', b') \wedge R_2(b', d') \wedge d'_j \neq c_j\} \\
S_j^{(3)} &= \{< a'_j, b''_j, d''_j > \ \mid P(a', b'') \wedge R_3(b'', d'') \wedge b''_j \neq b_j\}
\end{aligned}$$

Clearly, insertion of $P(a, b)$ causes violation since $< b, c > \in R$ and for any $j \in M$, $< a_j, b_j, c_j >$ cannot be in Q_j by construction of $S_j^{(1)}$, $S_j^{(2)}$ and $S_j^{(3)}$. To show there is no prior violation, we use the partition of R to show that no tuple in R contributes to a violation.

- $R_1(b', c')$: assume $\exists a'$ such that $P(a', b')$ holds. Since (41) is false, $a' \not\approx_X a$. There is some $j \in M$ such that $a'_j \neq a_j$. So $S_j^{(1)}(a'_j, b'_j, c'_j)$ hold.

- $R_2(b', d')$: since $d' \not\approx_Y c$, there is some $j \in M$ such that $d'_j \neq c_j$. So, if $\exists a'$ such that $P(a', b')$ holds, then $S_j^{(2)}(a'_j, b'_j, d'_j)$ must hold.
- $R_3(b'', d'')$: since $b'' \not\approx_Y b$, there is some $j \in M$ such that $b''_j \neq b_j$. So, if $\exists a'$ such that $P(a', b'')$ holds, then $S_j^{(3)}(a'_j, b''_j, d''_j)$ must hold.
- $R_4(b''', d''')$: since b''' has some c-components, there cannot be any matching tuple in P.

∎