Exploiting Dependencies to Enhance View Self-Maintainability

(Full Paper)

Nam Huyn
Stanford University
huyn@cs.stanford.edu

Abstract

View *self-maintenance* is the process of incrementally refreshing a materialized view using the view instance and the update to some base relation, but without examining any of the base relations, or using only a specified subset of the base relations. A new problem is created when base data use is limited in view maintenance: can the view be maintained at all? When a base relation is not available for use in maintaining a view, the next-best form of knowledge that may be available is *integrity constraints* the relation is known to satisfy. Constraints are often available in practice, yet very little is known about how they can be exploited fully to enhance view self-maintainability (SM). The problem becomes even harder when the SM tests are required to be in some efficient query form, e.g., SQL queries.

In this paper, we focus on the problem of determining view SM in the presence of functional dependencies. We first show that SM of a conjunctive-query (CQ) view can be reduced to a problem of query containment, whose solution can be expressed as a (boolean) query on the view in safe, nonrecursive Datalog$^{\neg\neq}$. We then show that for the special class of CQ views with no repeated predicates, two useful concepts can be defined: the *well-founded derivation DAG* and *subgoal partitioning.* We then derive three simple conditions (each expressible as the union of CQ's with $\neq$ comparisons) that each guarantee view SM under general functional dependencies. View maintenance expressions are shown to be simple CQ's. It turns out that these three conditions alone completely characterize view self-maintainability.

1 Introduction

We consider the *view self-maintenance* problem, that is, the problem of incrementally maintaining a materialized view when the base relations change, using the view itself and the update, but without using the base relations.

1.1 Motivation

Data warehouses have gained growing importance in recent years ([RED, IK93, Z*95]). One of their main advantages is providing materialized data against which a large number of queries can be executed cheaply, thus providing significant savings over mediated approaches where the queries instead have to go directly to the external data sources.

It can be argued that with data warehousing, the overall cost has not really been reduced much, but simply displaced from warehouse use to warehouse maintenance. There is, however, one difference. The availability of the warehouse itself as an extra data source for use to maintain the warehouse, can provide real opportunities to optimize maintenance cost, by minimizing the need to access external data sources.

Limited use of base data in view maintenance is therefore an important consideration in data warehousing. When a base relation is not available, maintaining a view may not always be possible, depending on the actual content of the view at the time the base data is updated. As a result, *runtime self-maintainability* (hereafter, just SM), the question of whether or not a given view can be maintained in the absence of some base relations, is a new but central question in maintenance. A base relation may not be available, but the next-best form of knowledge that is often available for free is *integrity constraints* the relation satisfies. Ignoring these constraints, while not catastrophic, often leads us to conclude that a view is not SM when actually it is, thus causing us to miss opportunities to save costly base data accesses. The main question this paper addresses is how to exploit constraints fully to enhance runtime view self-maintenance. This problem is not trivial, and its complexity is compounded by the fact that SM tests and the self-maintenance process must be relatively efficient themselves, or else, the savings that are expected from not using all the base data may be negated.

1.2 Historical Context

The problem of materialized view maintenance with unrestricted access to base relations has been well studied (see [GM95] for a comprehensive list of references). By contrast, limiting base data access opens up new dimensions to the maintenance problem which still remain largely unexplored. [GM95] also gave an excellent taxonomy for the different types of information available for view maintenance. In particular, the distinction between what we call runtime self-maintenance and compile-time self-maintenance is based on whether or not the contents of a given view and a given update are available to determine self-maintainability of the view under the update. In other words, while runtime self-maintainability guarantees that a view can be maintained using its own contents, under a specific view instance and a specific update, compile-time self-maintainability provides the same guarantee independent of the actual view instance and for all updates of a certain type. In this sense, the runtime approach is the more aggressive one; it may succeed where the compile-time approach would fail. While some works centered around compile-time self-maintenance have been written (such as [Q*95] on warehouse design and [GJM94] on maintenance), only a few works ([TB88, GB95]) are based on runtime self-maintenance, for which the issues of efficiency and constraints remain poorly understood.

1.3 Contribution of This Paper

The concept of runtime view self-maintenance is not new. The novelty of our work lies in using constraints to enhance runtime view self-maintainability and its maintenance, and in solving the problem with efficient maintainability tests and maintenance expressions. Specifically, we show that under functional dependencies, and for a view defined by a conjunctive query, we can generate SM tests in the form of efficient Datalog queries over the view. As such, not only can these tests be executed on traditional query engines, but we can also take advantage of any query optimization facility these engines might have, and any indices that might be defined on the view, to speed up their execution. For the special class of CQ views with no repeated predicates, we derive three conditions that, while much simpler than those obtained for the general case, completely characterize view SM under general functional dependencies.

1.4 Motivating Example

We begin with a simple example to illustrate the concept of runtime self-maintenance and to show how functional dependencies affect self-maintainability. The resulting SM tests are shown here without full explanation but will be more formally rederived in later sections. While this example conveys the salient points of the work, it does not reflect the full complexity of the problem of efficiently testing runtime SM under general functional dependencies.

Example 1.1 In its new marketing strategy to promote customer loyalty, a large chain store uses a data warehouse to collect customer purchase information, drawing on external data sources that may be its own operational databases or may belong to outside information brokers. The following source relations are used: $sales(Customer, Item, Store)$, collected from local branches, contains purchase transactions; $cust(Customer, Area, Bankcard)$, provided by a credit bureau, contains information about customers' place of residence and credit cards they possess; $comp(Rival, Item, Area)$, a customized database provided by an outside broker, indicates the presence of competing retailers in some geographic area together with the products they carry. A view V is defined by the query:

$$v(C, A, B, S, I, R) \;:\!-\; sales(C, I, S) \;\&\; cust(C, A, B) \;\&\; comp(R, I, A)$$

asking for customers who purchased some merchandise from the chain while the same merchandise can be bought from a competitor that has a presence in their residence area.

A new transaction $sales(cindy, igloo, springfield)$ is reported in. Since relations *cust* and *comp* can be accessed but only for a fee, the question is whether V can be updated without using these relations at all, that is, whether V is *self-maintainable* (SM). Suppose for a moment that we are totally ignorant about these relations, that is, no dependencies are known to hold among them. In this case, the most general condition that guarantees V to be SM (see [Hu96a]) is

$$v(cindy, -, -, -, igloo, -)$$

denoting the presence of some tuple in the view with the specific constants *cindy* and *igloo* in the C and I components respectively. Essentially, the presence of such tuple prevents the "adversary" from inventing a new area x that satisfies both $cust(cindy, x, -)$ and $comp(-, igloo, x)$, thus forcing V's update to depend on *cust* and *comp* by making it include tuple $(cindy, x, -, springfield, igloo, -)$.

Now suppose the data source guarantees that $Customer \rightarrow Area$ holds in *cust*. Intuitively, if we know *cindy*'s residence area, the adversary is no longer free to invent a different residence area for *cindy*, and the occurrence of both *cindy* and *igloo* in the same V tuple does not seem to be strictly needed to guarantee V's self-maintainability. Indeed, consider this particular view instance:

$$V = \{\ (cindy, a, b, s, ice, r)\ ,\ (carl, a, b', s', igloo, r')\ \}$$

On the one hand, to update V, we need to include at least $(cindy, a, b, springfield, igloo, r')$ in the insertion, since both $cust(cindy, a, b)$ and $comp(r', igloo, a)$ can be inferred to hold. On the other hand, for any tuple $(cindy, A, B, springfield, igloo, R)$ to be in V's update, A had better be a or else, *cindy* would have had two different places of residence. Furthermore, B had better be b or else, $cust(cindy, a, B)$ would have hold, and V would have contained $(cindy, a, B, s, ice, r)$ prior to the update. Similarly, R is identified with r'. Hence, the required view updates can be determined exactly without knowing the exact content of the base relations, and condition $v(cindy, -, -, -, igloo, -)$ is no longer necessary for the view to be SM. By ignoring functional dependencies, we have missed opportunities for saving base data accesses that may be costly. When the FD is taken into consideration, condition $v(cindy, -, -, -, igloo, -)$ can be replaced by the weaker condition

$$(\exists A)\ v(cindy, A, -, -, -, -) \wedge v(-, A, -, -, igloo, -)$$

which turns out to be the most general condition for SM given this FD. ∎

The challenges we tackle in this paper are summarized as follows. How do we reason with the given constraints in order to systematically generate, from a given view definition, tests that guarantee SM, along with view maintenance expressions. We are only interested in test conditions that are most general, that is, conditions that are both necessary and sufficient for SM. SM tests in query form are desirable, even more desirable are test queries that are simple and efficiently evaluable.

1.5 Paper Outline

The paper is organized as follows. Section 2 defines the notion self-maintainability, assumptions, terminology and notation used throughout the paper. Section 3 shows a reduction of SM to a query containment problem which can be solved with a safe, nonrecursive Datalog$^{\neg\neq}$ test query against the given view instance. For the special class of CQ views with no repeated predicates, Section 4 introduces the concepts of *well-founded derivation DAG* and *subgoal partitioning*. These concepts are used in Section 5 to derive three conditions that guarantees SM along with the corresponding maintenance expressions. These conditions (expressible as unions of CQ's over the view), while simpler than those obtained with the containment-based approach for the general case, completely characterize SM under functional dependencies. Related work appears in Section 6. Section 7 concludes with future work and open problems.

2 Preliminaries

In this paper, we are given a materialized view V. View V is defined by a given query $\mathcal{Q}$ posed against some database D (denoted $V = \mathcal{Q}(D)$). While the contents of the base relations from D are unknown, we are given a set $\mathcal{F}$ of functional dependencies that D satisfies (denoted $SAT(D, \mathcal{F})$). Let μ be an insertion to D. We assume μ is given, and D continues to satisfy the dependencies after the insertion. A database D is said to be *valid* (relative to the given V, $\mathcal{Q}$, $\mathcal{F}$ and insertion μ) if $\mathcal{Q}(D) = V$ and $SAT(D \cup \mu, \mathcal{F})$.

Definition 2.1 (Self-Maintainability): We say that view V is *self-maintainable* (SM) under the insertion of μ if the new view that results from the update does not depend on the underlying (valid) database, or more formally:

$$(\forall D_1, D_2)\ [\mathcal{Q}(D_1) = \mathcal{Q}(D_2) = V \wedge SAT(D_1 \cup \mu, \mathcal{F}) \wedge SAT(D_2 \cup \mu, \mathcal{F}) \Rightarrow \mathcal{Q}(D_1 \cup \mu) = \mathcal{Q}(D_2 \cup \mu)] \quad (1)$$

∎

Note that SM is defined with respect to a specific view V and a specific update μ. This definition assumes that no base relations are used to determine SM and to maintain the view.

Given V and μ, we will be looking for a condition on V and μ that is both necessary and sufficient for the view to be self-maintainable under the insertion. Such a condition will be expressed as a (boolean) query against view V, and will be referred to as a *SM test* or *test query*.

In the remainder of this paper, we assume the following:

- $\mathcal{Q}$ is a conjunctive query, defined the usual manner as in [Ull88]. The variables used in $\mathcal{Q}$'s body are assumed to also appear in $\mathcal{Q}$'s head.
- μ is a single insertion, and the updated predicate (i.e., the predicate whose relation is updated) is not repeated in $\mathcal{Q}$'s body. The inserted tuple is assumed to match the subgoal with the updated predicate since otherwise, the insertion cannot affect the view.
- All functional dependencies (FD's) have the form $\alpha \rightarrow \beta$ where α represents a set of attributes and β a single attribute.

For a CQ view V with the above restrictions, consider the special database instance, denoted $\mathcal{Q}^{-1}(V)$, that consists of all the ground atoms in $\mathcal{Q}$'s body that result from matching $\mathcal{Q}$'s head with every tuple in V. This database has the following property (details in Appendix A):

Proposition 2.1 *Provided that a valid database exists, $\mathcal{Q}^{-1}(V)$ is not only valid, but also minimal in the sense that it is contained in any valid database.* ∎

Example 2.1 Consider a view `V` defined by

$$\mathcal{Q}: \texttt{v(X,Y,Z) :- r(X,Y),s(X,Z),s(Y,Z).}$$

Suppose `V` $= \{$`(1,2,a)`, `(2,3,a)`$\}$. The first tuple generates the atoms `r(1,2)`, `s(1,a)` and `s(2,a)`. The second tuple generates `r(2,3)`, `s(2,a)` and `s(3,a)`. Thus $\mathcal{Q}^{-1}$(`V`) $= \{$`r(1,2)`, `r(2,3)`, `s(1,a)`, `s(2,a)`, `s(3,a)`$\}$ is the minimal valid database. Note that the relations in $\mathcal{Q}^{-1}$(`V`) can simply be defined by reversing the implication in the view definition. ∎

3 Testing Self-Maintainability based on Query Containment

In this section, we give a general method based on query containment (QC) for testing view SM under functional dependencies, thus providing an algorithm for generating tests that are themselves queries expressible in safe, nonrecursive Datalog$^{\neg\neq}$. We assume that the updated predicate may appear only once in $\mathcal{Q}$'s body, but the other predicates are allowed to occur more than once.

3.1 Reducing SM to QC

At first, the SM condition as defined in (1) does not appear to lend itself to a QC formulation which typically uses only one quantified database variable instead of two. A key idea consists of eliminating one of the two variables in (1), say D_1, by replacing it with some judiciously chosen canonical database instance $\hat{D}$, $\mathcal{Q}^{-1}(V)$ in our case. With many possible reductions still remaining to choose from, it is crucial to avoid using one that results in testing containment of CQ's involving negation, for which a closed-form solution is generally difficult to obtain. The choice of $\hat{D} \cup D$ for the other variable D_2, where D is a free variable, is another key idea. The following proposition can be used to reduce the SM problem to a problem of query containment.

Proposition 3.1 *Let $\mathcal{Q}$ be a CQ query over some database D, and let V be a view defined as $V = \mathcal{Q}(D)$. Consider the insertion of μ into D and let $\delta\mathcal{Q}_\mu(D)$ denote the set of tuples gained by $\mathcal{Q}$ as a result of adding μ to D. Then V is self-maintainable under the insertion of μ if and only if the following containment equation holds:*

$$(\forall D)\ \delta\mathcal{Q}_\mu(\hat{D} \cup D) \not\subseteq \delta\mathcal{Q}_\mu(\hat{D}) \Rightarrow \mathcal{Q}(\hat{D} \cup D) \not\subseteq V \vee \neg SAT(\hat{D} \cup D \cup \mu, \mathcal{F}) \tag{2}$$

∎

Proof: First, using the fact that $\hat{D}$ is a valid minimal database, we show that (1) is equivalent to

$$(\forall D)\ [\mathcal{Q}(\hat{D} \cup D) = V \wedge SAT(\hat{D} \cup D \cup \mu, \mathcal{F}) \Rightarrow \mathcal{Q}(\hat{D} \cup \mu) = \mathcal{Q}(\hat{D} \cup D \cup \mu)] \tag{3}$$

It is easy to see that (3) specializes (1) by substituting $\hat{D}$ for D_1 and $\hat{D} \cup D$ for D_2. Conversely, let D_1 and D_2 be two valid databases. Since $\hat{D}$ is contained in any valid database, there is some database D'_1 such that $D_1 = \hat{D} \cup D'_1$. Substituting D'_1 for D in (3), we derive $\mathcal{Q}(D_1 \cup \mu) = \mathcal{Q}(\hat{D} \cup \mu)$. Similarly, we can derive $\mathcal{Q}(D_2 \cup \mu) = \mathcal{Q}(\hat{D} \cup \mu)$. Thus $\mathcal{Q}(D_1 \cup \mu) = \mathcal{Q}(D_2 \cup \mu)$

Second, using the fact that $\mathcal{Q}$ is conjunctive and assuming that (3)'s left-hand-side holds, (3)'s right-hand-side can be rewritten as $V \cup \delta\mathcal{Q}_\mu(\hat{D}) = V \cup \delta\mathcal{Q}_\mu(\hat{D} \cup D)$, or equivalently, $\delta\mathcal{Q}_\mu(\hat{D} \cup D) \subseteq V \cup \delta\mathcal{Q}_\mu(\hat{D})$.

It is easy to verify that V can be dropped from the latter, using the fact that the updated predicate occurs only once in $\mathcal{Q}$. Equation (3) is now equivalent to the following:

$$(\forall D)\ [\mathcal{Q}(\hat{D} \cup D) = V \wedge SAT(\hat{D} \cup D \cup \mu, \mathcal{F}) \Rightarrow \delta\mathcal{Q}_\mu(\hat{D} \cup D) \subseteq \delta\mathcal{Q}_\mu(\hat{D})] \tag{4}$$

Finally, since $\mathcal{Q}$ is monotonic and $\hat{D}$ is valid, $\mathcal{Q}(\hat{D} \cup D) = V$ can be rewritten as $\mathcal{Q}(\hat{D} \cup D) \subseteq V$. We obtain (2) by considering the contrapositive formula. ∎

Example 3.1 Consider a view V defined by $\mathcal{Q} : v(X,Y,Z) \text{ :- } r(X,Y)\ \&\ s(X,Z)\ \&\ t(Z,Y).$, and the FD's $Y \rightarrow X$ on r, $X \rightarrow Z$ on s and $Z \rightarrow Y$ on t. Consider the insertion of $r(a,b)$. The canonical database $\hat{D}$ consists of predicates $\hat{r}$, $\hat{s}$ and $\hat{t}$ defined by the rules (one rule per subgoal in $\mathcal{Q}$, whose head is the subgoal, and whose body is $\mathcal{Q}$'s head):

$$\begin{array}{rcl} \hat{r}(X,Y) & \text{:-} & v(X,Y,Z). \\ \hat{s}(X,Z) & \text{:-} & v(X,Y,Z). \\ \hat{t}(Z,Y) & \text{:-} & v(X,Y,Z). \end{array}$$

$\delta\mathcal{Q}_\mu(\hat{D})$ consists all tuples (a,b,Z) where Z is generated by the rule:

$$u(Z) \text{ :- } \hat{s}(a,Z)\ \&\ \hat{t}(Z,b).$$

The following programs $\mathcal{P}_1$ and $\mathcal{P}_2$, with query predicate *panic*, implement the boolean conditions on the left and right hand sides respectively of containment equation (2):

$$\begin{array}{rrcl} \mathcal{P}_1 : & panic & \text{:-} & \hat{s}(a,Z)\ \&\ t(Z,b)\ \&\ \neg u(Z). \\ & panic & \text{:-} & s(a,Z)\ \&\ \hat{t}(Z,b)\ \&\ \neg u(Z). \\ & panic & \text{:-} & s(a,Z)\ \&\ t(Z,b)\ \&\ \neg u(Z). \end{array}$$

$$\begin{array}{rrclrcl} \mathcal{P}_2 : & panic & \text{:-} & \hat{r}(X,Y)\ \&\ \hat{s}(X,Z)\ \&\ t(Z,Y)\ \&\ \neg v(X,Y,Z). & panic & \text{:-} & \hat{r}(X',Y)\ \&\ r(X,Y)\ \&\ X \neq X'. \\ & panic & \text{:-} & \hat{r}(X,Y)\ \&\ s(X,Z)\ \&\ \hat{t}(Z,Y)\ \&\ \neg v(X,Y,Z). & panic & \text{:-} & r(X',Y)\ \&\ r(X,Y)\ \&\ X \neq X'. \\ & panic & \text{:-} & \hat{r}(X,Y)\ \&\ s(X,Z)\ \&\ t(Z,Y)\ \&\ \neg v(X,Y,Z). & panic & \text{:-} & r(X,b)\ \&\ X \neq a. \\ & panic & \text{:-} & r(X,Y)\ \&\ \hat{s}(X,Z)\ \&\ t(Z,Y)\ \&\ \neg v(X,Y,Z). & panic & \text{:-} & \hat{s}(X,Z')\ \&\ s(X,Z)\ \&\ Z \neq Z'. \\ & panic & \text{:-} & r(X,Y)\ \&\ s(X,Z)\ \&\ \hat{t}(Z,Y)\ \&\ \neg v(X,Y,Z). & panic & \text{:-} & s(X,Z')\ \&\ s(X,Z)\ \&\ Z \neq Z'. \\ & panic & \text{:-} & r(X,Y)\ \&\ s(X,Z)\ \&\ t(Z,Y)\ \&\ \neg v(X,Y,Z). & panic & \text{:-} & \hat{t}(Z,Y')\ \&\ t(Z,Y)\ \&\ Y \neq Y'. \\ & & & & panic & \text{:-} & t(Z,Y')\ \&\ t(Z,Y)\ \&\ Y \neq Y'. \end{array}$$

Thus, view V is self-maintainable under the insertion of $r(a,b)$ if and only if $\mathcal{P}_1 \subseteq \mathcal{P}_2$ for all instances of r, s and t, and for the particular instances of v, $\hat{r}$, $\hat{s}$, $\hat{t}$ and u fixed by the given view instance V. ∎

Generalizing from Example 3.1, the SM problem can be reduced to the problem of testing $\mathcal{P}_1 \subseteq \mathcal{P}_2$ for which the following facts can be stated (the details of the generalization can be found in Appendix B):

- Testing $\mathcal{P}_1 \subseteq \mathcal{P}_2$ involves finding containment mappings from $\mathcal{P}_2$ to $\mathcal{P}_1$. Since the updated predicate (r in Example 3.1) does not occur in $\mathcal{P}_1$, rules in $\mathcal{P}_2$ that use this predicate can be dropped. Hence, *the FD's on the updated relations have no effect on view self-maintainability.*
- Both $\mathcal{P}_1$ and $\mathcal{P}_2$ are unions of CQ's that use both predicates with known extensions (v, $\hat{r}$, $\hat{s}$, $\hat{t}$, u in Example 3.1) and predicates with unknown extensions (s, t in Example 3.1). The former are called *constant* predicates, the latter *variable* predicates. The conjunctive queries also involve $\neq$

comparisons and negation, but the subgoals under negation all use constant predicates and can be turned into $\neq$ comparisons. In fact, the constant symbols from all known predicate extensions can be *unfolded*, turning $\mathcal{P}_1$ and $\mathcal{P}_2$ into unions of CQ's with $\neq$ comparisons. Containment between such queries can be solved using known approaches such as [G*94, Klu88].

3.2 Testing query containment using the view instance

A rather subtle but important point needs to be made here. By unfolding the constant symbols from the known predicate extensions, we obtain expressions for the queries (and hence their containment tests) that use these constant symbols. For instance, if $p(\bar{X})$ is a subgoal with constant predicate p (whose extension P is totally known), the constant symbols from P can be unfolded by expanding the subgoal $p(\bar{X})$ into $\bigvee_{\bar{t}\in P} \bar{X} = \bar{t}$. Similarly the subgoal $\neg p(\bar{X})$ can be expanded into $\bigwedge_{\bar{t}\in P} \bar{X} \neq \bar{t}$. While these symbols are treated as constant in the context of the containment problem, they are just placeholders whose actual values are not determined until runtime. Thus, if we want to generate a query containment test that uses the view instance rather than the constant symbols contained in it, we must follow constants unfolding by a step that folds them back together into a predicate, as the following example illustrates.

Example 3.2 Consider the following CQ's whose rules are extracted from Example 3.1:

$$\begin{array}{ll} \mathcal{P}_1: & panic \;{:}{-}\; s(a,Z)\ \&\ t(Z,b)\ \&\ \neg u(Z). \\ \mathcal{P}_2: & panic \;{:}{-}\; \hat{r}(X,Y)\ \&\ s(X,Z)\ \&\ t(Z,Y)\ \&\ \neg v(X,Y,Z). \end{array}$$

Conjunctive query $\mathcal{P}_2$ can be rewritten as $\mathcal{P}_2'$, a union of the following conjunctive queries, with (x,y) ranging over tuples in the relation for $\hat{r}$:

$$panic \;{:}{-}\; s(X,Z)\ \&\ t(Z,Y)\ \&\ X = x\ \&\ Y = y\ \&\ \neg v(X,Y,Z).$$

Using an extension (presented in Appendix C) of the original result from [G*94] for testing containment of unions of CQ's with arithmetic comparisons, $\mathcal{P}_1 \subseteq \mathcal{P}_2$ holds if and only if:

$$(\forall Z)\ [\neg u(Z) \Rightarrow \bigvee_{(x,y)} \bigvee_{h} h(\ X = x \wedge Y = y \wedge \neg v(X,Y,Z)\ )]$$

where h ranges over all containment mappings from $\mathcal{P}_2'$ to $\mathcal{P}_1$. The mapping $h(X) = a$, $h(Y) = b$, $h(Z) = Z$ is not only the only mapping but is also independent of (x,y). To fold (x,y) back to the $\hat{r}$ predicate form, simply turn the disjunction $\bigvee_{(x,y)} \ldots$ into $(\exists x,y)\ \hat{r}(x,y)\ldots$ as follows:

$$(\forall Z)\ [\neg u(Z) \Rightarrow (\exists x,y)\ [\hat{r}(x,y) \wedge a = x \wedge b = y \wedge \neg v(a,b,Z)]]$$

∎

We have developed necessary and sufficient conditions for the containment of a class of queries that generalize those involved in (2): unions of conjunctive queries with negation and $\neq$ comparisons and using constant predicates. Our results build on the work in [G*94] by expressing the containment test as a first-order formula using the constant predicates. The details can be found in Appendix C. Let us just summarize the results with the following theorem.

Theorem 3.1 *Let v be the predicate for view V. There is a first-order formula $\tau_{v,\mu}$ using only v and μ that tests self-maintainability of V under the insertion of μ.* ∎

3.3 Making the test safe

Formula $\tau_{v,\mu}$ from Theorem 3.1 that tests $\mathcal{P}_1 \subseteq \mathcal{P}_2$ is unfortunately not safe in general. With $\bar{Z}$ denoting a set of variables, and $\bar{Z}_i$ a subset of $\bar{Z}$, $\tau_{v,\mu}$ generally takes the form (details in Appendix D):

$$(\forall \bar{Z})\ [\varphi_1(\bar{Z}_1) \wedge \varphi_2(\bar{Z}_2) \wedge \ldots \varphi_n(\bar{Z}_n) \Rightarrow \psi(\bar{Z})]$$

where $\psi(\bar{Z})$ represents a finite set of $\bar{Z}$-values, and $\varphi_i(\bar{Z}_i)$ a set of $\bar{Z}_i$-values that may not be finite. The obvious question is whether or not it is equivalent to a safe formula.

Example 3.3 Consider the following CQ's whose rules are extracted from Example 3.1:

$$\begin{array}{ll} \mathcal{P}_1: & panic \ :\!-\ s(a, Z)\ \&\ t(Z, b)\ \&\ \neg u(Z). \\ \mathcal{P}_2: & panic \ :\!-\ \hat{s}(X, Z')\ \&\ s(X, Z)\ \&\ Z \neq Z'. \end{array}$$

$(\forall Z)[\varphi(Z) \Rightarrow u(Z)]$ is the formula that tests $\mathcal{P}_1 \subseteq \mathcal{P}_2$, where $\varphi(Z)$ denotes $(\forall Z')[\hat{s}(a, Z') \Rightarrow Z = Z']$. Set $\varphi(Z)$ may or may not be finite, depending on $\hat{s}$: if $\hat{s}(a, Z')$ has no Z'-value, any Z would satisfy $\varphi(Z)$; otherwise, any Z that satisfies $\varphi(Z)$ must also satisfy $\hat{s}(a, Z)$. To make the formula safe, the main idea is to express $\varphi(Z)$ as conditional union of a finite set and an infinite set. Then, the formula, viewed as a set containment, becomes a conditional disjunction of two containments: a containment of two finite sets, and a containment of an infinite set in a finite set which is obviously false. ∎

The following theorem shows that the resulting containment tests can be practically implemented as efficient queries. The proof can be found in Appendix D.

Theorem 3.2 *Let $\tau_{v,\mu}$ be the first-order formula in Theorem 3.1 that tests self-maintainability of V under the insertion of μ and that was obtained with the containment-based method. $\tau_{v,\mu}$ can always be expressed as a safe, nonrecursive Datalog$^{\neg\neq}$ query that uses only v for its EDB predicates.* ∎

4 Well-Founded Derivation DAG's and Subgoal Partitioning

Section 3 essentially shows there are general algorithms based on testing query containment that can generate self-maintainability tests in the form of safe, nonrecursive Datalog$^{\neg\neq}$ queries. The test query that could be obtained for Example 3.1 is actually fairly complex. Yet, as we will show later, the same problem can solved with the much simpler test: $(\exists Y, Z)\ v(a, Y, Z)$. Reducing the complex test query to such a simple form appears to require using sophisticated optimization techniques. Yet, perhaps simpler SM tests could be obtained directly for some appropriately restricted class of views. For the rest of this paper, we consider the special class of CQ views all of whose predicates have single occurrence. This restriction allows us to derive SM test queries that are much simpler (unions of CQ's), as we will show in the next section. But first, let us introduce in this section the key concepts embodied in a direct approach that allows us to self-maintain a view more efficiently.

Let a view V be defined by a conjunctive query $\mathcal{Q}$ over some base relations that satisfy a set $\mathcal{F}$ of functional dependencies. We begin by claiming that in the context of determining self-maintainability of V under some insertion, no generality is lost if we assume $\mathcal{Q}$ is *rectified*, i.e., no constants and duplicate variables occur within each subgoal in $\mathcal{Q}$. If $\mathcal{Q}$ is not rectified, we can always rectify $\mathcal{Q}$ to obtain $\mathcal{Q}'$ and construct a set $\mathcal{F}'$ of FD's on $\mathcal{Q}'$ predicates such that V is self-maintainable under $\mathcal{F}$ when defined by $\mathcal{Q}$ if and only if it is self-maintainable under $\mathcal{F}'$ when defined by $\mathcal{Q}'$ (see Appendix E.1 for construction details). Consequently, $\mathcal{Q}$ will be represented throughout the remainder of this paper as:

$$v(\bar{X}, \bar{U}, \bar{Z})\ :\!-\ r(\bar{X}, \bar{U})\ \&\ S(\bar{U}, \bar{Z}).$$

where $\bar{X}$, $\bar{U}$ and $\bar{Z}$ denote disjoint (ordered) sets of variables, $r(\bar{X}, \bar{U})$ denotes the subgoal with the updated predicate, and $S(\bar{U}, \bar{Z})$ denotes a conjunction of subgoals with nonupdated predicates. Under the insertion of $r(\bar{x}, \bar{a})$, we are interested in evaluating the query $\{\bar{Z} : S(\bar{U}, \bar{Z}) \wedge \bar{U} = \bar{a}\}$ in order to determine the required updates to the view. For this reason, the variables in S are categorized into what we call *updated* variables ($\bar{U}$) and *private* variables ($\bar{Z}$).

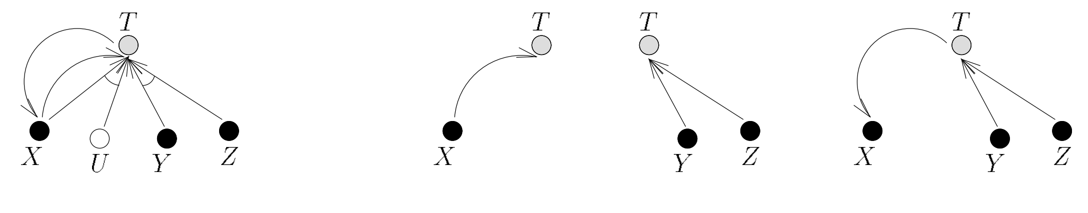
4.1 Well-Founded Derivation DAG's

FD's normally relate the attributes of a predicate. But since the subgoals in $\mathcal{Q}$ are rectified, we can think of the FD's as relating the variables in $\mathcal{Q}$, by further ignoring which predicates the original FD's apply to. Also note that the FD's originating from r can be ignored since they have no effect on SM (Subsection 3.1). The set of FD's on query variables can thus be represented by an AND/OR graph called the *dependency AND/OR graph*, where each node is associated with a unique variable in $\mathcal{Q}$. Of special interest are those (connected) AND acyclic subgraphs with a single sink node and all of whose source (resp. interior) nodes correspond to updated (resp. private) variables. We call these subgraphs *well-founded derivation DAG's.* The single sink node of a well-founded derivation DAG is also called the *root.* With functional dependencies represented as such, private variables are further categorized into determinable or nondeterminable. A private variable is said to be *determinable* when it corresponds to the root of some well-founded derivation DAG, *nondeterminable* otherwise.

Example 4.1 Consider the view definition:

$$v(X, Y, Z, T, U) \text{ :- } r(X, Y, Z) \ \& \ p(X, T, U) \ \& \ q(X, Y, Z, T).$$

and the FD's $XU \rightarrow T$ and $T \rightarrow X$ on p, and $X \rightarrow T$ and $YZ \rightarrow T$ on q. Using the notion of FD's on the view query variables, we simply say that X is determined by T, and T by either XU or YZ or X. The AND/OR graph that represents the dependencies between variables is depicted in Figure 1(a). Now, suppose r is the updated predicate. X, Y and Z are the updated variables, T and U private. All three well-founded derivation DAG's are shown in Figure 1(b), where the AND-connector that links an AND-node's incoming arcs is not shown. T is the only determinable variable, and U is nondeterminable. Updated nodes (nodes with an updated variable) are shown in Figure 1 in black, determinable nodes in grey, and nondeterminable nodes in white. ∎



(a) Dependency AND/OR graph (b) Well-founded derivation DAG's

Figure 1: Graphs of dependencies between query variables.

Intuitively, determinable variables are those $\bar{Z}$-variables in query $S(\bar{U}, \bar{Z})$ whose values are uniquely determined once the values of $\bar{U}$ are fixed, provided that the tuples that "instantiate" certain dependencies are known to be present in the base relations. We say that the presence of these tuples *forces* the determinable variables in the query $S(\bar{U}, \bar{Z})$ to agree on some specific values, making the query more specific. Also, we would like to point out that there are only a finite number of well-founded derivations DAG's, and there are algorithms (e.g. depth first) to extract them all.

4.2 Subgoal Partitioning

Central to view SM is the concept of *subgoal partitioning*, originally defined in [Hu96a] but extended here to handle functional dependencies. A set $S(\bar{V}, \bar{W})$ of subgoals is partitioned into groups that may not share any $\bar{W}$-variable, but such that any two subgoals from the same group must have some $\bar{W}$-variable in common. Formally, we define PART($S(\bar{V}, \bar{W})$,$\bar{W}$) to be the partitioning of S into groups $S_i(\bar{V}_i, \bar{W}_i)$ such that $\bar{V} = \bigcup_i \bar{V}_i$, $\bar{W} = \bigcup_i \bar{W}_i$, $\bar{W}_i$ and $\bar{W}_j$ are disjoint for $i \neq j$, and any pair of subgoals within group S_i must share some variable from $\bar{W}_i$.

Example 4.2 Consider the view definition from Example 4.1, and the subgoals (with nonupdated predicates) $p(X, T, U)$ and $q(X, Y, Z, T)$. While partition PART($\{p, q\}$,$\{T, U\}$) consists of the single group $\{p, q\}$, partition PART($\{p, q\}$,$\{U\}$) consists of the two groups $\{p\}$ and $\{q\}$. Metaphorically, variables $\{T, U\}$ (T in particular) in the first partition, and $\{U\}$ in the second one, act like glue that ties the subgoals together while we are trying to split them apart. Figure 2 illustrates the second partitioning where the glue $\{U\}$ fails to hold the subgoals together. ∎

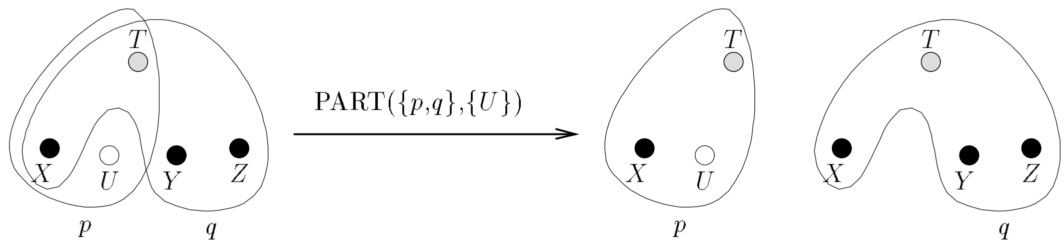


Figure 2: Subgoal partitioning.

The concept of subgoal partitioning will be applied to the view SM problem as follows. Consider the partitioning PART(S,$\bar{N}$) of the nonupdated subgoals that uses the nondeterminable variables $\bar{N}$ as "glue". We look for certain matching tuples in V that "conform" to PART(S,$\bar{N}$). This notion of "conform" will be made precise in a moment, but essentially the presence of such tuples assures that all required view updates can be computed from the view itself independently of the base relations.

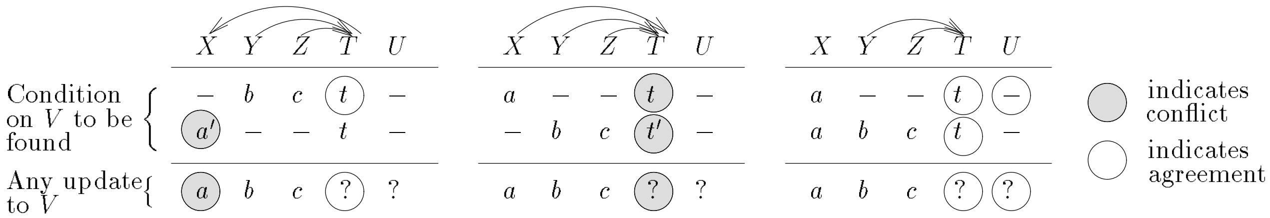
5 Efficient Tests for Self-Maintainability

There are many ways a view can be self-maintainable under a given insertion. Perhaps the simplest is when the view is not affected by the insertion. So we begin looking for a condition on the view instance that guarantees no tuples in the base relations can join with the inserted tuple $r(\bar{x}, \bar{a})$.

5.1 Forced Exclusion

Forced exclusion is a situation in which the presence of $S(\bar{a}, \bar{Z})$, the tuples that join with $r(\bar{x}, \bar{a})$, must be excluded in order to avoid conflicts due to the dependencies. The idea is to look for certain tuples in V that "instantiate" the dependencies in some well-founded derivation DAG's.

Example 5.1 Consider the view V and FD's in Example 4.1, and consider the insertion of $r(a, b, c)$. To update V on the one hand, any inserted tuple t_1 must be of the form (a, b, c, T, U). On the other hand, consider the set of dependencies $\{YZ \rightarrow T, T \rightarrow X\}$ that defines the well-founded derivation tree rooted at updated node X as shown in Figure 1(b), and suppose V contains some tuples $t_2 = (-, b, c, t, -)$ and $t_3 = (a', -, -, t, -)$ that "instantiate" these dependencies, where $a' \neq a$. If the updated view contains t_1, then in chasing the dependencies in the derivation tree bottom up, $YZ \rightarrow T$ forces t_1 and t_2 to agree on $T = t$, which in turn leads $T \rightarrow X$ to force t_1 and t_3 to agree on $a = a'$, hence leading to a contradiction. Thus the presence of both t_2 and t_3 in V excludes the presence of t_1, or else a conflict would be created over updated variable X, as illustrated in Figure 3(a). ∎



(a) Conflict over X (b) Conflict over T (c) Exposure of T and U

Figure 3: Conditions that guarantee V's updates to be independent of base relations.

Generalizing from Example 5.1, for each well-founded derivation DAG that is rooted at some updated node, we are looking for a set of tuples in V, one tuple for each dependency in the DAG, that instantiates the dependencies as follows: any tuple in the set agrees with the inserted tuple over the updated variables on the left hand side of its dependency; any pair of these tuples agrees over the determinable variables their dependencies may have in common; and the tuple for the root dependency disagrees with the inserted tuple over the root variable. Then $\mathcal{C}_{UPD}$, the disjunction of such conditions over all derivation DAG's rooted at some updated node, expresses the condition of forced exclusion that avoids conflicts over *updated* variables. The formula for $\mathcal{C}_{UPD}$, a union of CQ's, is shown in Appendix E.2.1. There is yet another situation in which the view cannot gain any new tuples, but based on conflicts over determinable variables.

Example 5.2 Continuing from Example 5.1, now consider the tuples $t_4 = (a, -, -, t, -)$ and $t_5 = (-, b, c, t', -)$ where $t \neq t'$, that instantiate the dependencies defining the two derivation trees commonly rooted at determinable node T shown in Figure 1(b). If the updated view contains t_1, then in chasing the dependencies in both trees, t_1 and t_4 are forced to agree on $T = t$, and t_1 and t_5 to agree on $T = t'$, leading to a contradiction again. Thus the presence of t_4 and t_5 in V also excludes the presence of t_1, but this time in order to avoid a conflict over determinable variable T (Figure 3(b)). ∎

Similarly, we can generalize from Example 5.2 to obtain $\mathcal{C}_{DET}$, the condition of forced exclusion that avoids conflicts over *determinable* variables, as follows: for each pair well-founded derivation DAG's that are commonly rooted at some determinable node, we look for tuples in V that instantiate the dependencies separately in each DAG, and such that the two tuples corresponding to the root dependencies disagree over the common root variable. $\mathcal{C}_{DET}$, a union of CQ's, is shown in Appendix E.2.2.

The following theorem, whose proof can be found in Appendices E.2.1 and E.2.2, shows that the view remains unchanged if the view and the insertion satisfy the two conditions above.

Theorem 5.1 *Let $\mathcal{C}_{UPD}$ and $\mathcal{C}_{DET}$ be the two conditions of forced exclusion as defined above for a given view, a given set of FD's on the base relations, and a given insertion. If either $\mathcal{C}_{UPD}$ or $\mathcal{C}_{DET}$ is satisfied, the insertion has no effect on the view.* ∎

5.2 Forced Exposure

We now turn to the case where view V may gain new tuples as a result of inserting $r(\bar{x}, \bar{a})$. *Forced exposure* is a situation in which the query $S(\bar{a}, \bar{Z})$ is forced to reveal the values for its private variables

$\bar{Z}$ through the view, thus making the required updates to the view entirely computable from the view itself. Two key ideas come into play here. First, in the absence of FD's, partition PART(S,$\bar{Z}$) allows us to generate the most general condition on V that forces all $\bar{Z}$-values in $S(\bar{a}, \bar{Z})$ to show up in the view ([Hu96a]), where PART(S,$\bar{Z}$) essentially treats $\bar{U}$ as the only "bound" variables. Second, in the presence of FD's, this idea generalizes in a surprisingly simple way: treat all determinable variables in $\bar{Z}$ as bound, in addition to the $\bar{U}$-variables. In other words, we consider the finer partition PART(S,$\bar{N}$), where $\bar{N}$ denotes the nondeterminable variables in $\bar{Z}$. Interestingly, the only aspect of FD's that counts is whether or not a variable is determinable.

Example 5.3 Consider the view V and FD's in Example 4.1, and consider the insertion of $r(a, b, c)$. As in Example 5.1, any update to V is a tuple t_1 of the form (a, b, c, T, U). As shown in Example 4.2, PART($\{p(X, T, U), q(X, Y, Z, T)\}$,$\{U\}$) contains two groups $\{p\}$ and $\{q\}$. Corresponding to the first group, consider looking for some tuple in V that agrees with the inserted tuple over the group's updated variables, that is, $t_6 = v(a, -, -, t, -)$. Similarly for the second group, consider tuple $t_7 = v(a, b, c, t, -)$. Note that t_6 and t_7 are also required to agree over the determinable variable(s) their groups may share, T in this case. While the presence of either t_6 or t_7 in V forces t_1 to agree on $T = t$, their simultaneous presence assures that t_1's remaining unknown U can be determined by looking up $v(a, -, -, t, U)$ (see Figure 3(c)), independently of relations p and q: when $t_6 \in V$, $(a, t, U) \in p$ if and only if $(a, -, -, t, U) \in V$, and when $t_7 \in V$, $(a, b, c, t) \in q$. The alert reader may notice that in this case, the tuples to be inserted into V are already in the view, but this fact does not hold in general. ∎

Generalizing from Example 5.3, the presence of tuples that "conform" to PART(S,$\bar{N}$), such as t_6 and t_7 in Example 5.3, can be formalized with $\mathcal{C}_A$, the condition of *forced exposure*, as follows:

$$\mathcal{C}_A: \quad (\exists \bar{d}) \bigwedge_g (\exists \bar{X}, \bar{U}, \bar{Z})\ v(\bar{X}, \bar{U}, \bar{Z}) \wedge \bar{U} =_{\bar{U}_g} \bar{a} \wedge \bar{Z} =_{\bar{D}_g} \bar{d}$$

where g ranges over the groups in PART($S(\bar{U}, \bar{D}, \bar{N})$,$\bar{N}$), $\bar{U}_g$ (resp. $\bar{D}_g$) denotes the updated (resp. determinable) variables used in group g, and $\bar{d}$ a vector of constants with the same dimension as $\bar{D}$. The fact that two vectors of constants $\bar{u}$ and $\bar{v}$ agree over variables $\bar{W}$ is denoted by $\bar{u} =_{\bar{W}} \bar{v}$.

The following theorem, whose proof can be found in Appendix E.2.3, shows that the condition of forced exposure is another way to guarantee view self-maintainability.

Theorem 5.2 *Let $\mathcal{C}_A$ be the condition of forced exposure as defined above for a view V, a set of FD's, and the insertion of $r(\bar{x}, \bar{a})$. If $\mathcal{C}_A$ is satisfied (say with constant $\bar{d}$), then the required updates to V exactly consist of inserting all tuples $(\bar{x}, \bar{a}, \bar{d}, \bar{n})$ such that the $\bar{n}_g$ component of $\bar{n}$ is determined by*

$$\{\bar{n}_g \mid (\exists \bar{X}, \bar{U}, \bar{Z})\ v(\bar{X}, \bar{U}, \bar{Z}) \wedge \bar{U} =_{\bar{U}_g} \bar{a} \wedge \bar{Z} =_{\bar{D}_g} \bar{d} \wedge \bar{Z} =_{\bar{N}_g} \bar{n}_g\},$$

where v is the predicate for V. These updates do not depend on the base relations. ∎

5.3 Complete Characterization of Self-Maintainability

Each of the three conditions $\mathcal{C}_A$, $\mathcal{C}_{UPD}$ and $\mathcal{C}_{DET}$ guarantees view SM. But are there other conditions that also provide that guarantee? In the following theorem, we claim that together, these three conditions completely characterize self-maintainability. The full proof is given in Appendix E.3.

Theorem 5.3 *Let $\mathcal{C}_{UPD}$ and $\mathcal{C}_{DET}$ be the two conditions of forced exclusion, and $\mathcal{C}_A$ the condition of forced exposure as defined above for a given view, a given set of FD's and a given insertion. The view is self-maintainable under the insertion if and only if ($\mathcal{C}_{UPD} \vee \mathcal{C}_{DET} \vee \mathcal{C}_A$).* ∎

Proof:(Sketch) The "If" part follows from Theorems 5.1 and 5.2. For the "Only If" part, assume $\mathcal{C}_A$, $\mathcal{C}_{UPD}$ and $\mathcal{C}_{DET}$ are all false. We need to find a counterexample that consists of two valid databases D_1 and D_2 that derive different views after the insertion. We choose D_1 to be $\mathcal{Q}^{-1}(V)$, and D_2 to be D_1 augmented with some set of tuples Δ constructed as follows. Δ initially includes some selected subset of $S(\bar{a}, \bar{D}, \bar{N})$, the choice being based on how $\mathcal{C}_A$ is falsified. The tuples in Δ may include variables. In chasing the FD's over D_2, some of these variables may be bound. The falsity of both $\mathcal{C}_{UPD}$ and $\mathcal{C}_{DET}$ assures that the chase process terminates without encountering a contradiction. Any variable that remains unbound is replaced with some new constant, and some selected tuples are removed from Δ. The falsity of $\mathcal{C}_A$ assures that D_2 is valid and that $\mathcal{Q}(D_2 \cup \mu) \neq \mathcal{Q}(D_1 \cup \mu)$. ∎

Example 5.4 Consider the view and FD's in Example 4.1. A necessary and sufficient condition for the view to be SM under the insertion of $r(a, b, c)$ is obtained by combining the conditions that characterize the presence of t_2 and t_3, t_4 and t_5, t_6 and t_7 introduced in Examples 5.1, 5.2 and 5.3:

$$(\exists T)\ [v(-, b, c, T, -) \wedge v(\not{a}, -, -, T, -)] \vee (\exists T)\ [v(-, b, c, T, -) \wedge v(a, -, -, \not{T}, -)] \vee v(a, b, c, -, -)$$

where $\not{a}$ denotes any value but a. The first and last disjuncts combine to simplify to $v(-, b, c, -, -)$ which completely subsumes the second disjunct. Hence the view is SM if and only if $v(-, b, c, -, -)$. ∎

Example 5.5 First consider the view in Example 3.1. Since there is only one derivation tree rooted at determinable Z, $\mathcal{C}_{DET}$ is vacuously false. Hence, the view is SM if and only if

$$(\exists Z)\ [v(a, -, Z) \wedge v(-, b, Z)] \vee (\exists Z)\ [v(a, -, Z) \wedge v(-, \not{b}, Z)]$$

This test simplifies to $(\exists Z)\ [v(a, -, Z) \wedge v(-, -, Z)]$ or just $v(a, -, -)$. Now consider the view defined in Example 1.1 with FD *Customer*$\rightarrow$ *Area*, and consider the insertion of *sales*(*cindy*, *igloo*, *springfield*). As there is only one derivation tree and its root A is determinable, the SM test degenerates to $(\exists A)\ v(cindy, A, -, -, -, -) \wedge v(-, A, -, -, igloo, -)$, since both $\mathcal{C}_{UPD}$ and $\mathcal{C}_{DET}$ are vacuously false. ∎

6 Related Work

The problem of view self-maintainability was originally studied in [TB88] and more recently in [GB95] for views that are conjunctive queries with arithmetic comparisons but all of whose predicates have single occurrence. Although [TB88] and [GB95] gave necessary and sufficient conditions for *conditionally autonomously computable updates* (a notion very similar to self-maintainability), these conditions use the constant symbols contained in the extension of the view. As such, they are not easily expressible as queries over the view. Constraints on the base relations were not considered, and efficient implementation of SM remains difficult with their approach. [Q*95] solves a problem different from but related to ours, in which a view is made (compile-time) self-maintainable by *designing auxiliary views* to materialize. In constructing the auxiliary views, [Q*95] exploits key and referential integrity constraints but not functional dependencies in their full generality. In [GJM94] the compile-time SM problem was addressed, rather than the run-time SM problem considered in this paper, and only key constraints were considered. [Hu96a] solved the SM problem with efficient test queries, but constraints were not considered. Finally, we believe that the way dependencies are exploited to solve the run-time SM problem is new and interesting, as opposed to solving traditional query containment and optimization problems addressed in previous works such as [ASU79, JK84, Sag87].

7 Conclusion

We have developed two methods to solve the runtime view self-maintenance problem under functional dependencies: a method based of testing query containment and a direct method based on the concepts of well-founded derivation DAG and subgoal partitioning. While the first method appears to be more general and extensible (e.g. when additional views are available to determine SM), it introduces complexity in the SM tests that requires sophisticated techniques to optimize away. In constrast, the direct method has the advantage of being more intuitive and more amenable to optimizations. In particular, redundancies among the FD's can be removed as in computing a minimal cover ([Ull88]), without affecting the validity of our results, thus making the SM test queries even simpler. The dependencies that relate the view query variables are in fact FD's over the view, and it should be of no suprise that the SM problem could benefit from this type of optimization. However, with general CQ views and general FD's, how much further optimization one can expect to achieve is still an open problem. Aside from a few obvious special cases known to admit very simple solutions (e.g. when no private variables are determined by only updated variables), research is under way to identify common classes of FD's and/or CQ views that admit even simpler SM tests. Another direction we are currently pursuing is to extend our results to views that do not contain all body variables and to handle other types of updates and dependencies.

References

[ASU79] Aho A. V., Sagiv Y., Ullman J. D.: Efficient optimization of a class of relational expressions. In *ACM Trans. Database Systems,* 4(1979), pp. 435–454.

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

[GJM94] Gupta A., Jagadish H. V. and Mumick I. S.: Data Integration Using Self-Maintainable Views. In *Proc. 5th EDBT Conf.,* Avigon, France, March 1966.

[GM95] Gupta A. and Mumick I. S.: Maintenance of Materialized Views: Problems, Techniques, and Applications. In *IEEE Data Engineering Bulletin,* 18(2), June 1995.

[G*94] Gupta A., Sagiv Y., Ullman J. D. and Widom J.: Constraint Checking with Partial Information. In *Proc. 13th ACM Symp. on Principles of Database Systems,* 1994, pp. 45–55.

[Hu96a] Huyn N.: Efficient View Self-Maintenance. In *Proc. Workshop on Materialized Views: Techniques and Applications,* pp. 17–25, 1996. Also avail. as `http://www-db.stanford.edu/pub/papers/cqvsm.ps` .

[IK93] Inmon W. H. and Kelley C.: Rdb/VMS: Developing the Data Warehouse. *QED Publishing Group,* Boston, Massachussetts, 1993.

[JK84] Johnson D. S., Klug A.: Testing Containment of Conjunctive Queries under Functional and Inclusion Dependencies. In *Journal of Computer and System Sciences,* 28(1979), pp. 167–189.

[Klu88] Klug, A.: On conjunctive queries containing inequalities. In *J. ACM* **35**:1, pp. 146–160, 1988.

[Q*95] Quass D., Gupta A., Mumick I. and Widom J.: Making Views Self-Maintainable for Data Warehousing. In *Parallel and Distributed Information Systems,* 1996.

[RED] Red Brick Systems. *Red Brick Warehouse,* 1995.

[Sag87] Sagiv Y.: Optimizing datalog programs. In *Proc. 6th ACM Symp. on PODS,* 1987, pp. 349–362.

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

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

[Z*95] Zhuge Y., Garcia-Molina H., Hammer J. and Widom J: View Maintenance in a Warehousing Environment. In *Proc. ACM SIGMOD,* San Jose, California, 1995.

A Canonical Databases

Let V be a view defined as $\mathcal{Q}(D)$ for some database D. $\mathcal{Q}$ is assumed to be a conjunctive query whose body variables all appear in the query's head. D is assumed to satisfy a given set $\mathcal{F}$ of functional dependencies. Consider the insertion of some tuple μ into D. We assume that database D continues to satisfy $\mathcal{F}$ after the insertion of μ.

A database D' is said to be valid (relative to V, $\mathcal{Q}$, $\mathcal{F}$, and μ) when $\mathcal{Q}(D') = V$ and D' satisfies $\mathcal{F}$ both before and after the insertion of μ. An example of valid database is D.

Let $\hat{D}$ be the canonical database $\mathcal{Q}^{-1}(V)$ formed by collecting all the ground atoms in $\mathcal{Q}$'s body that result from matching $\mathcal{Q}$'s head with every tuple in V.

First, it is easy to see that $\hat{D}$ is contained in any valid database D', since $\mathcal{Q}(D') = V$ and for every tuple $t \in V$, the ground atoms that result from matching $\mathcal{Q}$'s head with t must belong to D'. In other words, D' contains all atoms in $\hat{D}$.

Second, $\hat{D}$ itself is valid. We first show that $\mathcal{Q}(\hat{D}) = V$. It is clear that $\mathcal{Q}(\hat{D}) \supseteq V$ since $\hat{D}$ contains all the atoms that contribute to every tuple in V. Furthermore, since $\hat{D} \subseteq D$ and $\mathcal{Q}$ is monotonic, we conclude that $\mathcal{Q}(\hat{D}) \subseteq \mathcal{Q}(D) = V$. We then show that $\hat{D} \cup \mu$ satisfies $\mathcal{F}$. Since $D \cup \mu$ satisfies $\mathcal{F}$, any subset, $\hat{D} \cup \mu$ in particular, also satisfies $\mathcal{F}$.

Therefore, $\mathcal{Q}^{-1}(V)$ is the unique minimal valid database.

B Self-Maintainability as Containment of Conjunctive Queries

Consider a view V defined by the following conjunctive query

$$\mathcal{Q} : v(\bar{X}, \bar{Y}, \bar{Z}) \ :\!-\ r(\bar{X}, \bar{Y}), s_1(\bar{Y}_1, \bar{Z}_1), \ldots, s_n(\bar{Y}_n, \bar{Z}_n).$$

where $r(\bar{X}, \bar{Y})$ is the only subgoal whose relation is updated, $\bar{Y} = \bigcup_{i=1}^{n} \bar{Y}_i$, and $\bar{Z} = \bigcup_{i=1}^{n} \bar{Z}_i$. Predicates may be used more than once among the s_i subgoals, and each s_i may use constants.

Containment equation (2), which characterizes the condition under which V is self-maintainable under the insertion of $r(\bar{a}, \bar{b})$, can be elaborated as follows.

Database $\hat{D}$, which consists of predicate $\hat{r}$ and the predicates for the s_i subgoals, can be defined by the following rules:

$$\begin{aligned} \hat{r}(\bar{X}, \bar{Y}) \ &:\!-\ v(\bar{X}, \bar{Y}, \bar{Z}). \\ \hat{s}_i(\bar{Y}_i, \bar{Z}_i) \ &:\!-\ v(\bar{X}, \bar{Y}, \bar{Z}). \end{aligned}$$

For any subset $I \subseteq \{1, \ldots, n\}$, define S_I to be the result of replacing some of the predicates that occur among the s_i subgoals with their counterparts in $\hat{D}$ as follows:

$$S_I(\bar{Y}, \bar{Z}) \ :\!-\ \bigwedge_{i \in I} s_i(\bar{Y}_i, \bar{Z}_i), \bigwedge_{j \notin I} \hat{s}_j(\bar{Y}_j, \bar{Z}_j).$$

$\delta\mathcal{Q}_\mu(\hat{D} \cup D)$ consists all tuples (a, b, Z) where Z is generated by $S_I(\bar{b}, \bar{Z})$ (I taken over all subsets of $\{1, \ldots, n\}$). $\delta\mathcal{Q}_\mu(\hat{D})$ consists all tuples (a, b, Z) where Z is generated by $S_\emptyset(\bar{b}, \bar{Z})$.

On the one hand, the boolean query on the left of (2) is expressed by the following program $\mathcal{P}_1$ using *panic* for its query predicate (I taken over all subsets of $\{1, \ldots, n\}$):

$$panic \ :\!-\ S_I(\bar{b}, \bar{Z}), \neg S_\emptyset(\bar{b}, \bar{Z}). \tag{5}$$

On the other hand, the boolean query on the right of (2) is expressed by the following program $\mathcal{P}_2$ (with I taken over all subsets of $\{1,\ldots,n\}$, $i=1,\ldots,n$, and $\alpha \rightarrow \beta$ an FD on the appropriate predicate):

$$
\begin{array}{lll}
panic & :- & \hat{r}(\bar{X},\bar{Y}), S_I(\bar{Y},\bar{Z}), \neg v(\bar{X},\bar{Y},\bar{Z}). \quad (6)\\
panic & :- & r(\bar{X},\bar{Y}), S_I(\bar{Y},\bar{Z}), \neg v(\bar{X},\bar{Y},\bar{Z}). \quad (7)\\
panic & :- & \hat{r}(\bar{T}), r(\bar{T}'), \bar{T} =_\alpha \bar{T}', \bar{T} \neq_\beta \bar{T}'. \quad (8)\\
panic & :- & r(\bar{T}), r(\bar{T}'), \bar{T} =_\alpha \bar{T}', \bar{T} \neq_\beta \bar{T}'. \quad (9)\\
panic & :- & r(\bar{T}), \bar{T} =_\alpha \bar{a}\bar{b}, \bar{T} \neq_\beta \bar{a}\bar{b}. \quad (10)\\
panic & :- & \hat{s}_i(\bar{T}_i), s_i(\bar{T}'_i), \bar{T}_i =_\alpha \bar{T}'_i, \bar{T}_i \neq_\beta \bar{T}'_i. \quad (11)\\
panic & :- & s_i(\bar{T}_i), s_i(\bar{T}'_i), \bar{T}_i =_\alpha \bar{T}'_i, \bar{T}_i \neq_\beta \bar{T}'_i. \quad (12)
\end{array}
$$

Solving $\mathcal{P}_1 \subseteq \mathcal{P}_2$ involves finding containment mappings from $\mathcal{P}_2$ to $\mathcal{P}_1$. Since predicate r does not occur in $\mathcal{P}_1$, rules in $\mathcal{P}_2$ that use r can be dropped, namely (7), (8), (9) and (10). Hence, the FD's on r are not relevant to the SM problem, as stated in the following proposition.

Proposition B.1 *FD's on the updated relation have no effect on view self-maintainability.* ▮

C Solving the QC problem

The queries we want to compare both consist of rules with the same head (*panic*) and whose body contains positive subgoals with both "constant" predicates and "variable" predicates, negated subgoals with constant predicates, and $=$ and $\neq$ comparisons. A *constant* predicate refers to a predicate whose relation is totally known (either because it is given or derived from other known relations). If the relation for a predicate is totally unknown, the predicate is said to be *variable*.

The constant predicates can be eliminated by replacing any subgoal with such a predicate by a disjunction of equality comparisons involving the symbols in the relation for the constant predicate. Thus, if p is a constant predicate with relation P, the subgoal $p(\bar{X})$ can be expanded to $\bigvee_{\bar{t}\in P} \bar{X} = \bar{t}$, and $\neg p(\bar{X})$ to $\bigwedge_{\bar{t}\in P} \bar{X} \neq \bar{t}$. By eliminating all constant predicates and by factoring out all the disjunctions, a CQ of the form we are interested here can be rewritten as a union of CQ's with only variable predicates but also with inequality comparisons. First, recall the following result from [G*94] paraphrased here:

Theorem C.1 **([G*94])** *Consider the conjunctive queries $\mathcal{Q}_i$: H_i:–B_i, A_i where B_i represents a conjunction of ordinary subgoals and A_i a conjunction of arithmetic comparisons. B_i is assumed not to use the same variable twice or any constant. Let M be the set of containment mappings from (H_2,B_2) to (H_1,B_1). Then $\mathcal{Q}_1 \subseteq \mathcal{Q}_2$ if and only if $A_1 \Rightarrow \bigvee_{h\in M} h(A_2)$.* ▮

Note that if a conjunction of ordinary subgoals uses the same variable twice or uses constants, we can always *rectify* it (that is, eliminate all constants and duplicate variables) by introducing enough new variables and equate these new variables with the constants and duplicate variables. In the case of Theorem C.1, A_i would include all equality comparisons that result from the rectification of B_i.

Even though Theorem C.1 can be used to decide the containment of our queries, the problem is that using this straightforward approach leads to containment test expressions involving symbols from the relation for the constant predicates, but these symbols are not known at view definition time. We would rather like to have test expressions that refer to the given relations as a whole rather than their contents, as is usually the case with traditional database queries.

The solution is to use the relation contents of the constant predicates only as a conceptual tool and to develop containment tests that actually use these constant predicates in their *unexpanded* form. To this end, we now extend the containment results from [G*94] to deal symbolically with constant predicates. The results are stated in the following lemmas.

Lemma C.1 *Consider the queries* $\mathcal{P} : G \text{:-} R, \bigvee_i A_i$ *and* $\mathcal{Q} : H \text{:-} S, \bigvee_j B_j$*, where* R *and* S *represent conjunctions of ordinary subgoals that do not use the same variable twice or any constant,* A_i *and* B_j *represent conjunctions of arithmetic comparisons. Let* M *be the set of containment mappings from* (H,S) *to* (G,R)*. Then* $\mathcal{P} \subseteq \mathcal{Q}$ *if and only if* $(\bigvee_i A_i) \Rightarrow \bigvee_{h \in M} h(\bigvee_j B_j)$*.* ∎

Proof: The queries we want to compare are just unions of conjunctive queries with arithmetic comparisons. Applying Theorem C.1 (paraphrased from [G*94]), the containment holds if and only for every i, the following holds

$$A_i \Rightarrow \bigvee_j \bigvee_{h \in M} h(B_j)$$

The disjunction over j can be pushed all the way into the B_j's. Furthermore, since the right hand side of each of these implications is independent of i, the A_i's can be regrouped into a disjunction. ∎

Lemma C.2 *Consider the queries* $\mathcal{Q}_i : H_i \text{:-} B_i, A_i$ *where* B_i *represents a conjunction of ordinary subgoals and* A_i *a boolean combination of arithmetic comparisons.* B_i *is assumed not to use the same variable twice or any constant. Let* M *be the set of containment mappings from* (H_2,B_2) *to* (H_1,B_1)*. Then* $\mathcal{Q}_1 \subseteq \mathcal{Q}_2$ *if and only if* $A_1 \Rightarrow \bigvee_{h \in M} h(A_2)$*.* ∎

Proof: Lemma C.2 follows from Lemma C.1 since a boolean combination of arithmetic comparisons can always be expressed as a disjunction of conjunctions of arithmetic comparisons. ∎

Using Lemma C.2, we can derive two theorems that deal with queries of different forms. These theorems require some set of subgoals to be rectified. A set of subgoals is said to be *rectified* when no variables occur more than once among the subgoals, and no constant symbols are used. Rectifying a set of subgoals simply involves introducing new variables and introducing additional subgoals that equate the new variables with constants or existing variables.

The first theorem characterizes the containment of CQ's containing both positive and negative subgoals with constant predicates, and applies to rules such as (6).

Theorem C.2 *Consider the queries* $\mathcal{Q}_i : H_i(\bar{X}'_i, \bar{Y}'_i, \bar{Z}'_i) \text{:-} A_i(\bar{X}_i, \bar{Y}_i), B_i(\bar{Y}_i, \bar{Z}_i), \neg c_i(\bar{X}_i, \bar{Y}_i, \bar{Z}_i)$ *where* A_i *represents a conjunction of subgoals with constant predicates,* B_i *a conjunction of subgoals with variable predicates and* c_i *an atom with a constant predicate. Let* M *be the set of containment mappings from* $(H_2,$*rectified* $B_2)$ *to* (H_1,B_1)*. Let* $G_h(\bar{Y}_1, \bar{Z}_1)$ *be the result of applying some* $h \in M$ *to the equality comparisons obtained from the rectification of* B_2*. Then* $\mathcal{Q}_1 \subseteq \mathcal{Q}_2$ *if and only if:*

$$\begin{aligned}(\forall \bar{X}_1, \bar{Y}_1, \bar{Z}_1) \;\; & A_1(\bar{X}_1, \bar{Y}_1) \wedge \bigwedge_{h \in M} [\neg G_h(\bar{Y}_1, \bar{Z}_1) \vee [(\forall \bar{X}_2) \; A_2(\bar{X}_2, h(\bar{Y}_2)) \Rightarrow c_2(\bar{X}_2, h(\bar{Y}_2), h(\bar{Z}_2))]] \\ & \Rightarrow c_1(\bar{X}_1, \bar{Y}_1, \bar{Z}_1) \end{aligned} \tag{13}$$

∎

Proof: The proof essentially consists of putting the given queries in an appropriate form so that Lemma C.2 can be applied. In fact, an extension of the lemma will be used instead, where: (1) only the second query needs rectification; (2) the second query is a union of subqueries, in which case the implicant in the containment condition needs to be augmented with a disjunction over each subquery.

Query $\mathcal{Q}_1$ already has the required form, since: $A_1(\bar{X}_1, \bar{Y}_1)$ can be viewed as a disjunction of conjunctions of equality comparisons involving variables in $\bar{X}_1$ and $\bar{Y}_1$ on the one hand, and constant symbols from known relations on the other hand; similarly, $\neg c_1(\bar{X}_1, \bar{Y}_1, \bar{Z}_1)$ can be viewed as a conjunction of disjunctions of inequality comparisons involving variables in $\bar{X}_1$, $\bar{Y}_1$ and $\bar{Z}_1$ on the one hand, and constant symbols from known relations on the other hand.

Query $\mathcal{Q}_2$ can be viewed as a union, over each tuple $(\bar{x}_2, \bar{y}_2) \in A_2$, of the following subquery:

$$H_2(\bar{X}'_2, \bar{Y}'_2, \bar{Z}'_2) \text{:-} B_2(\bar{Y}_2, \bar{Z}_2), (\bar{Y}_2 = \bar{y}_2), \neg c_2(\bar{x}_2, \bar{y}_2, \bar{Z}_2)$$

which, after rectification, takes the form:

$$H_2(\bar{X}'_2, \bar{Y}'_2, \bar{Z}'_2) \text{:-} D_2(\bar{Y}_2, \bar{Z}_2, \bar{N}), E(\bar{N}, \bar{Y}_2, \bar{Z}_2), (\bar{Y}_2 = \bar{y}_2), \neg c_2(\bar{x}_2, \bar{y}_2, \bar{Z}_2)$$

where $\bar{N}$ represents new variables introduced, D_2 has no duplicate variables or constants, and E represents the additional equality comparisons introduced by the rectification.

Applying the extension of Lemma C.2 as mentioned above to $\mathcal{Q}_1$ and the modified $\mathcal{Q}_2$ (a union of subqueries), the containment holds if and only if:

$$\begin{aligned}(\forall \bar{X}_1, \bar{Y}_1, \bar{Z}_1) \quad & A_1(\bar{X}_1, \bar{Y}_1) \wedge \neg c_1(\bar{X}_1, \bar{Y}_1, \bar{Z}_1) \\ & \Rightarrow \bigvee_{(\bar{x}_2, \bar{y}_2) \in A_2} \bigvee_{h \in M} h[E(\bar{N}, \bar{Y}_2, \bar{Z}_2) \wedge (\bar{Y}_2 = \bar{y}_2) \wedge \neg c_2(\bar{x}_2, \bar{y}_2, \bar{Z}_2)]\end{aligned}$$

where M is the set of mappings from (H_2, D_2) to (H_1, B_1) that map variables in $\bar{Y}_2$, $\bar{Z}_2$, $\bar{N}$ to variables in $\bar{Y}_1$, $\bar{Z}_1$ and constants. Note that neither M nor E depends on $\bar{x}_2$ and $\bar{y}_2$. We use this fact to push the outer disjunction inside and turn the disjunction into an existential statement, to obtain the following for the implicant:

$$\bigvee_{h \in M} h[E(\bar{N}, \bar{Y}_2, \bar{Z}_2) \wedge (\exists\ \bar{x}_2, \bar{y}_2)(A_2(\bar{x}_2, \bar{y}_2) \wedge (\bar{Y}_2 = \bar{y}_2) \wedge \neg c_2(\bar{x}_2, \bar{y}_2, \bar{Z}_2))]$$

By defining $G_h(\bar{Y}_1, \bar{Z}_1)$ to be $h[E(\bar{N}, \bar{Y}_2, \bar{Z}_2)$, and by replacing $\bar{y}_2$ with $h(\bar{Y}_2)$, we obtain:

$$\bigvee_{h \in M} G_h(\bar{Y}_1, \bar{Z}_1) \wedge (\exists\ \bar{x}_2)(A_2(\bar{x}_2, h(\bar{Y}_2)) \wedge \neg c_2(\bar{x}_2, h(\bar{Y}_2), h(\bar{Z}_2)))$$

∎

The second theorem, which applies to queries that check for FD violations (e.g. rules (11), (12)), can be similarly derived.

Theorem C.3 *Consider the queries:*

- $\mathcal{P}_1 : H_1 \text{:-} A_1(\bar{X}_1, \bar{Y}_1), B_1(\bar{Y}_1, \bar{Z}_1), \neg c_1(\bar{X}_1, \bar{Y}_1, \bar{Z}_1).$, *where A_1 represents a conjunction of subgoals with constant predicates, B_1 a conjunction of subgoals with variable predicates and c_1 an atom with a constant predicate.*

- $\mathcal{P}_2 : H_2 \text{:-} A_2(X_2, \bar{Y}_2), B_2(X'_2, \bar{Y}_2, \bar{Z}_2), X_2 \neq \bar{X}'_2.$, *where A_2 represents a subgoal with constant predicate, and B_2 a subgoal with variable predicate (and that is already rectified).*

- $\mathcal{P}_3 : H_3$:-$A_3(X_3, \bar{Y}_3, \bar{Z}_3), B_3(X_3', \bar{Y}_3', \bar{Z}_3'), \bar{Y}_3 = \bar{Y}_3', X_3 \neq \bar{X}_3'$., *where both* A_3 *and* B_3 *represent a subgoal with variable predicate, (and that is already rectified).*

Then $\mathcal{P}_1 \subseteq \mathcal{P}_2$ *if and only if:*

$$(\forall \bar{X}_1, \bar{Y}_1, \bar{Z}_1) A_1(\bar{X}_1, \bar{Y}_1) \wedge \bigwedge_h (\forall X_2)\ [A_2(X_2, h(\bar{Y}_2')) \Rightarrow X_2 = h(X_2')] \Rightarrow c_1(\bar{X}_1, \bar{Y}_1, \bar{Z}_1) \tag{14}$$

where h *ranges over all containment mappings from* (H_2, B_2) *to* (H_1, B_1)*, and* $\mathcal{P}_1 \subseteq \mathcal{P}_3$ *if and only if:*

$$(\forall \bar{X}_1, \bar{Y}_1, \bar{Z}_1) A_1(\bar{X}_1, \bar{Y}_1) \wedge \bigwedge_h [h(\bar{Y}_3) = h(\bar{Y}_3') \Rightarrow h(X_3) = h(X_3')] \Rightarrow c_1(\bar{X}_1, \bar{Y}_1, \bar{Z}_1) \tag{15}$$

where h *ranges over all containment mappings from* (H_3, A_3, B_3) *to* (H_1, B_1)*,* ∎

Both theorems can be easily extended the obvious way to handle unions. Thus, the original SM problem reduces to a QC problem that can be solved using a test in the form of a first order query over known relations.

D Making Certain Implication Formulas Safe

In some cases, the problem of finding complete tests for view self-maintainability can be solved using query containment techniques. For at least conjunctive-query views, the question of self-maintainability can be reduced to a question of containment of union of conjunctive queries with inequality comparisons. Solving such containment problems often results in first-order-test formulas that are generally not safe. For example, (13), (14), and (15) are such formulas where the variables in $\bar{Z}_1$ are not range-restricted. Since it is desirable to implement these tests as queries that can be evaluated on conventional query engines, we are lead to wonder whether the test formulas can always be made safe (e.g., rewritten in safe, nonrecursive Datalog$^{\neg\neq}$).

The following notation convention will be used throughout the rest of this section.

- Predicates with name starting with p, q, r, s, t will be used to represent safe queries and finite relations
- Queries that are not necessarily safe are represented with predicates with name starting with a, b, c, d, e.
- $\bar{Z}$ denotes an ordered set of variables. $\bar{Z}_i$, $\bar{Z}'$ denote subsets of $\bar{Z}$, $\bar{Z}_i'$ denotes a subset of $\bar{Z}_i$.

The formulas in question are of the form

$$(\forall \bar{Z})\ a_1(\bar{Z}_1) \wedge a_2(\bar{Z}_2) \wedge \ldots \wedge a_n(\bar{Z}_n) \Rightarrow r(\bar{Z}) \tag{16}$$

where r represents a safe query and $a_i(\bar{Z}_i)$ denotes a (generally unsafe) query having one of the following forms:

- $\alpha_i(\bar{Z}_i)$: $(\forall \bar{X}_i)\ p_i(\bar{X}_i) \Rightarrow q_i(\bar{X}_i, \bar{Z}_i)$, where p_i and q_i represent safe queries.
- $\beta_i(\bar{Z}_i)$: $(\forall \bar{X}_i)\ p_i(\bar{X}_i, \bar{Z}_i') \Rightarrow q_i(\bar{X}_i, \bar{Z}_i)$ where $\bar{Z}_i' \subseteq \bar{Z}_i$ and $\bar{Z}_i' \neq \emptyset$, and p_i and q_i represent safe queries.
- $\gamma_i(\bar{Z}_i)$: $\neg[(\mu_1 = \nu_1) \wedge \ldots \wedge (\mu_m = \nu_m)]$, where the μ_j's and ν_j's are either constants or variables from $\bar{Z}_i$. We assume that no comparison involves the same variable.

- $\delta_i(\bar{Z}_i)$: of the form $(\forall X_i)\ p_i(\bar{Z}_i, X_i) \Rightarrow, X_i = q_i$, or $(\forall X_i)\ p_i(\bar{Z}'_i, X_i) \Rightarrow, X_i = Z''_i$, where p_i denotes a safe query, q_i is a constant, $\bar{Z}'_i \subseteq \bar{Z}_i$, and $Z''_i \in \bar{Z}_i$.

Formula (16) is generally not safe and the question of whether or not it can always be made safe is not immediately obvious. In the following, we will show that formula (16) can always be rewritten as a safe (boolean) query: we will first give a lemma that deal with the α_i subqueries, and another lemma that deal with the β_i and γ_i subqueries. The two lemmas are then combined to show the main theorem.

Lemma D.1 *Let I be some set of indices and let $b(\bar{Z}')$ denote some arbitrary (not necessarily safe) query. Formula $(\forall \bar{Z})\ (\bigwedge_{i\in I} \alpha_i(\bar{Z}_i)) \wedge b(\bar{Z}') \Rightarrow r(\bar{Z})$ can be rewritten as the disjunction over all subsets $H \subseteq I$ of*

$$(\bigwedge_{i\in H} s_i) \wedge (\bigwedge_{i'\in I-H} \neg s_{i'}) \wedge (\forall \bar{Z})\ [(\bigwedge_{i\in H} t_i(\bar{Z}_i)) \wedge b(\bar{Z}') \Rightarrow r(\bar{Z})]$$

where s_i and $t_i(\bar{Z}_i)$ are safe queries. ∎

Proof: Let s_i be the safe (boolean) query $(\exists \bar{X}_i)\ p_i(\bar{X}_i)$, and let $t_i(\bar{Z}_i)$ be the safe query $(\exists \bar{X}_i)\ q_i(\bar{X}_i, \bar{Z}_i) \wedge \alpha_i(\bar{Z}_i)$. Since $\alpha_i(\bar{Z}_i)$ is logically equivalent to $(\neg s_i \vee t_i(\bar{Z}_i))$, the premise of the implication can be expanded and rewritten as the disjunction over all subsets $H \subseteq I$ of

$$(\bigwedge_{i\in H} s_i) \wedge (\bigwedge_{i'\in I-H} \neg s_{i'}) \wedge (\bigwedge_{i\in H} t_i(\bar{Z}_i)) \wedge b(\bar{Z}')$$

The original formula now has the form $(\forall \bar{Z})\ [\bigvee_H (s_H \wedge t_H(\bar{Z}_H) \wedge b(\bar{Z}'))] \Rightarrow r(\bar{Z})$, which is equivalent to

$$\bigwedge_H [s_H \Rightarrow (\forall \bar{Z})\ (t_H(\bar{Z}_H) \wedge b(\bar{Z}') \Rightarrow r(\bar{Z}))]$$

Since the s_H's are mutually exclusive, we can rewrite the latter formula as

$$\bigvee_H [s_H \wedge (\forall \bar{Z})\ (t_H(\bar{Z}_H) \wedge b(\bar{Z}') \Rightarrow r(\bar{Z}))]$$

∎

Lemma D.2 *Let J and K be two disjoint sets of indices and let $p(\bar{Z}')$ denote a safe query. For any $H \subseteq J$, let $\bar{Z}'_H$ denotes $\bar{Z}' \cup \bigcup_{j\in H} \bar{Z}_j$. When $\bar{Z}' \subset \bar{Z}$, formula $(\forall \bar{Z})\ p(\bar{Z}') \wedge (\bigwedge_{j\in J} \beta_j(\bar{Z}_j)) \wedge (\bigwedge_{k\in K} \gamma_k(\bar{Z}_k)) \Rightarrow r(\bar{Z})$ can be rewritten as the conjunction over all $H \subseteq J$ of either*

$$(\forall \bar{Z})\ [p(\bar{Z}') \wedge (\bigwedge_{j\in H} t_j(\bar{Z}_j)) \wedge (\bigwedge_{j'\in J-H} \neg s_{j'}(\bar{Z}'_{j'})) \wedge (\bigwedge_{k\in K} \gamma_k(\bar{Z}_k)) \Rightarrow r(\bar{Z})]$$

for those H such that $\bar{Z}'_H = \bar{Z}$, or

$$\neg(\exists \bar{Z}'_H)\ [p(\bar{Z}') \wedge (\bigwedge_{j\in H} t_j(\bar{Z}_j)) \wedge \bigwedge_{j'\in J-H, \bar{Z}'_{j'} \subseteq \bar{Z}'_H} \neg s_{j'}(\bar{Z}'_{j'}) \wedge \bigwedge_{k\in K, \bar{Z}_k \subseteq \bar{Z}'_H} \gamma_k(\bar{Z}_k)]$$

for those H such that $\bar{Z}'_H \subset \bar{Z}$, where the $s_j(\bar{Z}'_j)$'s and $t_j(\bar{Z}_j)$'s are safe queries. ∎

Proof: Let $s_j(\bar{Z}'_j)$ be $(\exists \bar{X}_j)\ p_j(\bar{X}_j, \bar{Z}'_j)$ and $t_j(\bar{Z}_j)$ be $[(\exists \bar{X}_j)\ q_j(\bar{X}_j, \bar{Z}_j)] \wedge \beta_j(\bar{Z}_j)$. Rewrite $\beta_j(\bar{Z}_j)$ as $(t_j(\bar{Z}_j) \vee \neg s_j(\bar{Z}'_j))$, that is, the union of a finite set and a cofinite set. The premise of the original formula expands to a disjunction, and the formula itself expands to a conjunction (over H) of implications of the form

$$(\forall \bar{Z})\ [p(\bar{Z}') \wedge (\bigwedge_{j \in H} t_j(\bar{Z}_j)) \wedge (\bigwedge_{j' \in J-H} \neg s_{j'}(\bar{Z}'_{j'})) \wedge (\bigwedge_{k \in K} \gamma_k(\bar{Z}_k)) \Rightarrow r(\bar{Z})] \tag{17}$$

If $Z'_H = Z$, (17) is safe and we are done. Otherwise, if $Z'_H \subset Z$, consider the condition

$$(\exists \bar{Z}'_H)\ [p(\bar{Z}') \wedge (\bigwedge_{j \in H} t_j(\bar{Z}_j)) \wedge \bigwedge_{j' \in J-H, \bar{Z}'_{j'} \subseteq \bar{Z}'_H} \neg s_{j'}(\bar{Z}'_{j'}) \wedge \bigwedge_{k \in K, \bar{Z}_k \subseteq \bar{Z}'_H} \gamma_k(\bar{Z}_k)] \tag{18}$$

If there is a value $\bar{z}'_H$ satisfying (18), there would be infinitely many ways to extend $\bar{z}'_H$ to the remaining variables in $\bar{Z}$ to satisfy the premise in (17), since each of the remaining conjuncts in (17)'s premise uses a variable not in $\bar{Z}'_H$. But since r is finite, (17) cannot be true. Conversely, if (18) is false, the premise in (17) is not satisfiable and (17) is vacuously true. ∎

Theorem D.1 *Formula $(\forall \bar{Z})\ a_1(\bar{Z}_1) \wedge \ldots \wedge a_n(\bar{Z}_n) \Rightarrow r(\bar{Z})$, where $a_i(\bar{Z}_i)$ is either of the form $\alpha_i(\bar{Z}_i)$, $\beta_i(\bar{Z}_i)$ or $\gamma_i(\bar{Z}_i)$, can always be rewritten as a safe query.* ∎

Proof: For some I, J and K, the original formula can be rewritten as

$$(\forall \bar{Z})\ (\bigwedge_{i \in I} \alpha_i(\bar{Z}_i)) \wedge (\bigwedge_{j \in J} \beta_j(\bar{Z}_j)) \wedge (\bigwedge_{k \in K} \gamma_k(\bar{Z}_k)) \Rightarrow r(\bar{Z}) \tag{19}$$

Applying Lemma D.1, (19) can be rewritten as the disjunction over all subsets $H \subseteq I$ of $(\bigwedge_{i \in H} s_i) \wedge (\bigwedge_{i' \in I-H} \neg s_{i'}) \wedge \varphi_H$, where φ_H is defined as

$$(\forall \bar{Z})\ (\bigwedge_{i \in H} t_i(\bar{Z}_i)) \wedge (\bigwedge_{j \in J} \beta_j(\bar{Z}_j)) \wedge (\bigwedge_{k \in K} \gamma_k(\bar{Z}_k)) \Rightarrow r(\bar{Z})] \tag{20}$$

Let $\bar{Z}_H = \bigcup_{i \in H} \bar{Z}_i$. If $\bar{Z}_H = \bar{Z}$, since the subquery $(\bigwedge_{i \in H} t_i(\bar{Z}_i))$ in (20) is safe, (20) is safe and we are done. Otherwise, if $\bar{Z}_H \subset \bar{Z}$, we can apply Lemma D.2 to rewrite (20) as a safe query. ∎

Using similar a technique, Theorem D.1 can be extended to handle the inclusion of the δ_i form.

E Self-Maintainability of CQ Views with no Repeated Base Predicates

E.1 Rectified Views

Let a view V be defined by query $\mathcal{Q}$ over some base relations that satisfy a set $\mathcal{F}$ of functional dependencies. $\mathcal{Q}$ is assumed to be a conjunctive query where no base predicates occur more than once and all body variables appear in the query's head.

The following can be used to rectify $\mathcal{Q}$ by constructing the $\mathcal{Q}'$ and $\mathcal{F}'$ that are equivalent to the original query and FD's for the purpose of determining SM of V.

When a subgoal g in $\mathcal{Q}$ uses some constant or uses some variable more than once, the corresponding subgoal g' in $\mathcal{Q}'$ is obtained from g by using a new predicate and dropping the constants and all copies of variables. If a subgoal g in $\mathcal{Q}$ is already rectified, the corresponding subgoal g' in $\mathcal{Q}'$ is g unmodified.

For every subgoal g in $\mathcal{Q}$, the following rules determine, for every FD $\alpha \rightarrow \beta$ on the predicate of g, the FD on the predicate for the corresponding g' in $\mathcal{Q}'$.

- Case 1: if β is equated in g to a constant, or if β and some attribute in α are equated in g, ignore $\alpha \rightarrow \beta$.
- Case 2: otherwise, eliminate any attribute in α that is equated to a constant in g and combine any pair of attributes that are equated in g.

Proposition E.1 *Under single insertions, a view V is self-maintainable under $\mathcal{F}$ when defined by $\mathcal{Q}$ if and only if it is self-maintainable under $\mathcal{F}'$ when defined by $\mathcal{Q}'$.* ▮

Proof: Given the base relation R for some subgoal g in $\mathcal{Q}$, we construct the corresponding base relation R' for $\mathcal{Q}'$ by simply keeping those tuples from R that satisfy g and projecting out those attributes that have constants or duplicated variables in g. Conversely, given some base relation R' for $\mathcal{Q}'$, we construct the corresponding base relation R for $\mathcal{Q}$ by simply extending R' with additional attributes and filling these attributes with constants or copies of other components. Let D denote the database for $\mathcal{Q}$, and D' the corresponding database for $\mathcal{Q}'$. The construction above specifies D' given D, and D given D'.

First, the way D' is constructed from D does not affect the query result. In other words, if D satisfies $\mathcal{Q}(D) = V$, then the corresponding D' satisfies $\mathcal{Q}'(D') = V$. For similar arguments, it is easy to see that the converse also holds.

Second, we show that if D satisfies $\mathcal{F}$, then the corresponding D' also satisfies $\mathcal{F}'$. To see why, consider an FD $\alpha' \rightarrow \beta' \in \mathcal{F}'$, and let t_1' and t_2' be two tuples from the same relation in D'. Tuples t_1' and t_2' derive from some tuples t_1 and t_2 respectively from D. Also, since only Case 2 above can generate FD's for D', $\alpha' \rightarrow \beta'$ must derive from some FD $\alpha \rightarrow \beta'$ from $\mathcal{F}$. Assume that t_1' and t_2' agree over α'. It follows that t_1 and t_2 also agree over α'. Since D satisfies $\mathcal{F}$, t_1 and t_2 must agree over β', and hence t_1' and t_2' agree over β'. Conversely, we show that if D' satisfies $\mathcal{F}'$, then the corresponding D also satisfies $\mathcal{F}$. For FD's that fall into Case 2 above, the proof is analoguous to that for the converse above. We only need to show that D satisfies any FD that falls into Case 1. Let t_1 and t_2 be two tuples from the same relation R in D, and $\alpha \rightarrow \beta$ be an FD over that relation, and let g be the corresponding subgoal in $\mathcal{Q}$. For the subcase (of Case 1) where β is equated in g to a constant, all tuples from R, and t_1 and t_2 in particular, have that constant for their β components. For the subcase where β and some attribute in α are equated in g, if t_1 and t_2 agree over α, then they must also agree over β since they both satisfy g.

Finally, we prove the proposition using the definition of self-maintainability and the observations above. ▮

E.2 The Three Conditions Are Sufficient

In this section, we consider the insertion of $r(\bar{x}, \bar{a})$.

E.2.1 Avoiding Conflicts over Updated Variables

Let $\mathcal{D}$ a well-founded derivation DAG whose root node (the sink of the DAG) is a updated variable. Let $\alpha_{\mathcal{D}} \rightarrow \beta_{\mathcal{D}}$ be the FD at the root node, $\bar{D}_{\mathcal{D}}$ denote the variables at the internal nodes, and $DEP_{\mathcal{D}}$ denote the set of FD's at all $\mathcal{D}$'s internal nodes.

Consider the condition:

$$\mathcal{C}_{UPD} : \bigvee_{\mathcal{D}} (\exists \bar{D}_{\mathcal{D}}) \; [\; (\exists \bar{X}, \bar{U}, \bar{Z})(v(\bar{X}, \bar{U}, \bar{Z}) \wedge \bar{Z} =_{\alpha_{\mathcal{D}}} \bar{D}_{\mathcal{D}} \wedge \bar{U} =_{\alpha_{\mathcal{D}}} \bar{a} \wedge \bar{U} \neq_{\beta_{\mathcal{D}}} \bar{a})$$
$$\wedge \bigwedge_{\alpha \rightarrow \beta \in DEP_{\mathcal{D}}} (\exists \bar{X}, \bar{U}, \bar{Z})(v(\bar{X}, \bar{U}, \bar{Z}) \wedge \bar{Z} =_{\alpha} \bar{D}_{\mathcal{D}} \wedge \bar{U} =_{\alpha} \bar{a} \wedge \bar{U} =_{\beta} \bar{a})]$$

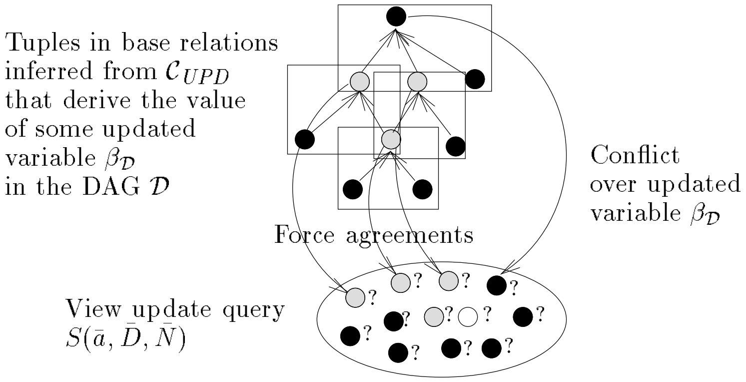


Figure 4: A derivation-DAG view of computing required view updates.

where the disjunction ranges over all DAG's with an updated root. Essentially, $\mathcal{C}_{UPD}$ looks for a well-founded derivation of a value for some updated variable that disagrees with $\bar{a}$ over that variable. We claim that when evaluated to true, this condition guarantees not only that the required update to the view does not depend on the base relations, but also that view does not require any update. The claim is stated in the following lemma.

Lemma E.1 *Let V be a view, $\mathcal{F}$ a set of FD's over the base relations, and $\mathcal{C}_{UPD}$ the condition specified above. If $\mathcal{C}_{UPD}$ is satisfied, the insertion of $r(\bar{x},\bar{a})$ has no effect on the view.* ∎

Proof: (Sketch) Let us rewrite $\mathcal{C}_{UPD}$ as $\bigvee_{\mathcal{D}}(\exists \bar{D}_{\mathcal{D}})\theta_{\mathcal{D}}(\bar{D}_{\mathcal{D}})$. Intuitively, when there is a derivation DAG $\mathcal{D}$ and a constant value $\bar{d}_{\mathcal{D}}$ that satisfy $\theta_{\mathcal{D}}(\bar{D}_{\mathcal{D}})$, there are no other values that also satisfy the latter condition since all the nodes' values are uniquely determined by the source nodes of the $\mathcal{D}$. Furthermore, $\theta_{\mathcal{D}}(\bar{D}_{\mathcal{D}})$ assures the presence of certain tuples in the base relations that force the variables $\bar{D}_{\mathcal{D}} \subseteq \bar{D}$ in query $S(\bar{a}, \bar{D}, \bar{N})$ to agree with $\bar{d}_{\mathcal{D}}$ and also force disagreement at the root node, as depicted in Figure 4. Thus, the query $S(\bar{a}, \bar{D}, \bar{N})$ has no possible answer. ∎

E.2.2 Avoiding Conflicts over Determinable Variables

Consider the condition:

$$
\begin{aligned}
\mathcal{C}_{DET}: \bigvee_{(\mathcal{D}_1,\mathcal{D}_2)} (\exists X_1, X_2)\ [\ &(\exists \bar{D}_{\mathcal{D}_1})[(\exists \bar{X}, \bar{U}, \bar{Z})(v(\bar{X}, \bar{U}, \bar{Z}) \wedge \bar{Z} =_{\alpha_{\mathcal{D}_1}} \bar{D}_{\mathcal{D}_1} \wedge \bar{U} =_{\alpha_{\mathcal{D}_1}} \bar{a} \wedge \bar{Z} =_{\beta_{\mathcal{D}_1}} X_1) \\
&\qquad \wedge \bigwedge_{\alpha \to \beta \in DEP_{\mathcal{D}_1}} (\exists \bar{X}, \bar{U}, \bar{Z})(v(\bar{X}, \bar{U}, \bar{Z}) \wedge \bar{Z} =_{\alpha\beta} \bar{D}_{\mathcal{D}_1} \wedge \bar{U} =_{\alpha} \bar{a})] \\
&\wedge (\exists \bar{D}_{\mathcal{D}_2})[(\exists \bar{X}, \bar{U}, \bar{Z})(v(\bar{X}, \bar{U}, \bar{Z}) \wedge \bar{Z} =_{\alpha_{\mathcal{D}_2}} \bar{D}_{\mathcal{D}_2} \wedge \bar{U} =_{\alpha_{\mathcal{D}_2}} \bar{a} \wedge \bar{Z} =_{\beta_{\mathcal{D}_2}} X_2) \\
&\qquad \wedge \bigwedge_{\alpha \to \beta \in DEP_{\mathcal{D}_2}} (\exists \bar{X}, \bar{U}, \bar{Z})(v(\bar{X}, \bar{U}, \bar{Z}) \wedge \bar{Z} =_{\alpha\beta} \bar{D}_{\mathcal{D}_2} \wedge \bar{U} =_{\alpha} \bar{a})] \\
&\wedge X_1 \neq X_2]
\end{aligned}
$$

where the disjunction ranges over all pairs of DAG's sharing the same determinable root.

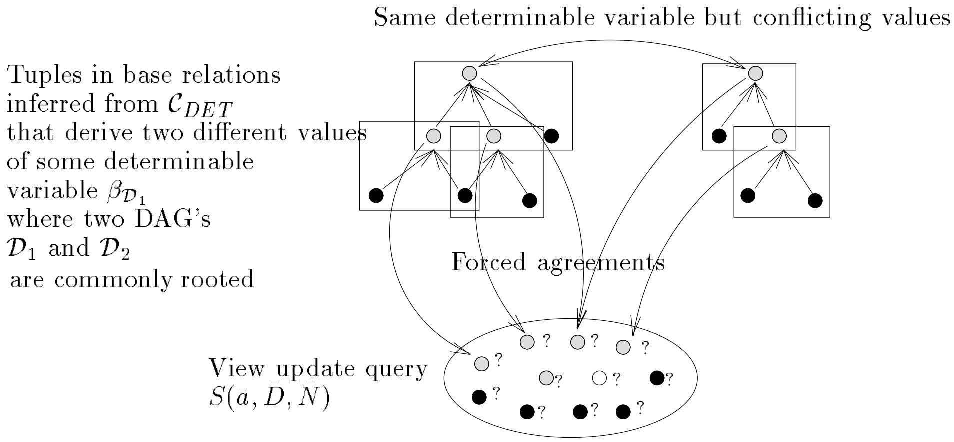


Figure 5: Two derivation DAG's disagreeing on their common root.

Essentially, $\mathcal{C}_{DET}$ looks for two different well-founded derivations for some determinable variable that yields different values. We claim that when evaluated to true, this condition guarantees not only that the required update to the view does not depend on the base relations, but also that view does not require any update. The claim is stated in the following lemma.

Lemma E.2 *Let V be a view, $\mathcal{F}$ a set of FD's over the base relations, and $\mathcal{C}_{DET}$ the condition specified above. If $\mathcal{C}_{DET}$ is satisfied, the insertion of $r(\bar{x},\bar{a})$ is guarantee to cause no change to V.* ∎

Proof: (Sketch) Intuitively, the condition assures the presence of certain tuples in the base relations that forces some determinable variable in the query $S(\bar{a}, \bar{D}, \bar{N})$ to agree with two conflicting values, each derived through a different derivation, as depicted in Figure 5. Hence, the query $S(\bar{a}, \bar{D}, \bar{N})$ has no possible answer. ∎

E.2.3 Forcing Exposure

In this section, we give a proof sketch for Theorem 5.2.

First, when $\mathcal{C}_A$ is satisfied with some constant value $\bar{d}$ for variable $\bar{D}$, there are no other values that would satisfy it, since any value for $\bar{D}$ is uniquely determined by $\bar{a}$. This claim can be substantiated by considering every derivation DAG rooted at some determinable node and by showing that all interior nodes in the DAG are uniquely determined (starting from the deepest node and ending at the root node).

Furthermore, $\mathcal{C}_A$ assures the presence of certain tuples in the base relations that force $\bar{D}$ in query $S(\bar{a}, \bar{D}, \bar{N})$ to agree with $\bar{d}$. This statement can be shown using arguments similar to showing the uniqueness of $\bar{d}$. The situation is illustrated in Figure 6.

So let $\bar{d}$ the unique value that satisfies $\mathcal{C}_A$. We show that the query $\mathcal{P}(\bar{N}) : \{\bar{N} \mid S(\bar{a}, \bar{d}, \bar{N})\}$, which determines the required updates to the view, can be computed from the view alone independently of the base relations. Consider the partition PART($S(\bar{U}, \bar{D}, \bar{N})$,$\bar{N}$). Since the groups in the partition do not share any N-variables, $\mathcal{P}(\bar{N})$ is entirely determined if $S_g(\bar{a}_g, \bar{d}_g, \bar{N}_g)$ can be computed for every group g. Now we claim that

$$\{\bar{n}_g \mid S_g(\bar{a}_g, \bar{d}_g, \bar{n}_g)\} = \{\bar{n}_g \mid (\exists \bar{X}, \bar{U}, \bar{Z})\ v(\bar{X}, \bar{U}, \bar{Z}) \wedge \bar{U} =_{\bar{U}_g} \bar{a} \wedge \bar{Z} =_{\bar{D}_g} \bar{d} \wedge \bar{Z} =_{\bar{N}_g} \bar{n}_g\}$$

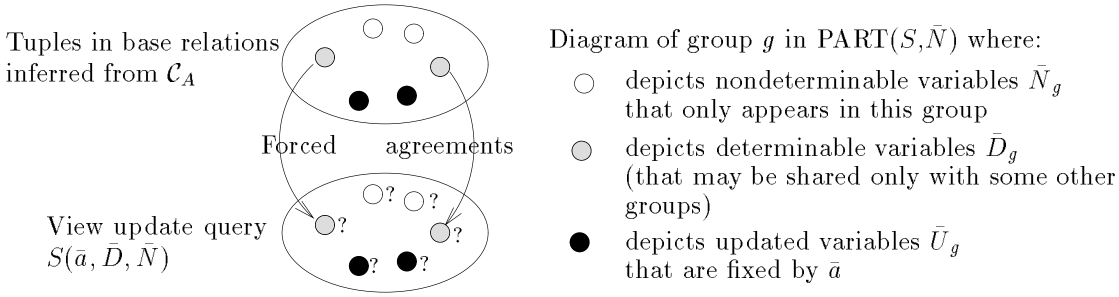


Figure 6: A subgoal-partition view of computing required view updates.

The "$\supseteq$" part of the claim is obvious since the only way to explain the presence in V of a tuple that agrees with $\bar{a}$, $\bar{d}$, and $\bar{n}_g$ over variables $\bar{U}_g$, $\bar{D}_g$, and $\bar{N}_g$ respectively, is the necessary presence of all the atoms in $S_g(\bar{a}_g, \bar{d}_g, \bar{n}_g)$.

For the "$\subseteq$" part of the claim, the satisfaction of $(\exists\bar{X}, \bar{U}, \bar{Z})v(\bar{X}, \bar{U}, \bar{Z}) \wedge \bar{U} =_{\bar{U}_g} \bar{a} \wedge \bar{Z} =_{\bar{D}_g} \bar{d}$ assures the presence of a set of tuples (one in each base relation not used in S_g) that join with $S_g(\bar{a}_g, \bar{d}_g, \bar{n}_g)$, for any $\bar{n}_g$. The result of this join will appear among the tuples in the view that agree with $\bar{a}$, $\bar{d}$, and $\bar{n}_g$ over variables $\bar{U}_g$, $\bar{D}_g$, and $\bar{N}_g$ respectively. In other words, any $\bar{n}_g$ that satisfies $S_g(\bar{a}_g, \bar{d}_g, \bar{n}_g)$ will appear in the set on the right hand of the equality we are trying to prove.

E.3 The Three Conditions Are Necessary

In the following, we show that the disjunction of $\mathcal{C}_A$, $\mathcal{C}_{UPD}$, and $\mathcal{C}_{DET}$ is necessary for self-maintainability. We will consider the partitioning of $S(\bar{U}, \bar{D}, \bar{N})$ at various levels of granularity: the coarser partition PART(S,$\bar{D} \cup \bar{N}$), the finer partition PART(S,$\bar{N}$), and some partition in between to be specified later. The coarser partition decomposes S into groups $g_i : S_i(\bar{U}_i, \bar{D}_i, \bar{N}_i)$. The finer partition can be alternatively viewed as decomposing each g_i into the smaller groups $g_{ij} : S_{ij}(\bar{U}_{ij}, \bar{D}_{ij}, \bar{N}_{ij})$. In this notation, we write $\mathcal{C}_A$ as

$$\bigwedge_i (\exists \bar{d}_i) \bigwedge_j (\exists \bar{X}, \bar{U}, \bar{Z})\ v(\bar{X}, \bar{U}, \bar{Z}) \wedge \bar{U} =_{\bar{U}_{ij}} \bar{a} \wedge \bar{Z} =_{\bar{D}_{ij}} \bar{d}_i$$

Note that $\mathcal{C}_A$, the condition that guarantees the presence of tuples that "conform" to PART(S,$\bar{N}$), subsumes the condition that guarantees the presence of tuples that "conform" to the coarser partition PART(S,$\bar{D} \cup \bar{N}$). The latter condition can be written as $\bigwedge_i (\exists \bar{X}, \bar{U}, \bar{Z})\ v(\bar{X}, \bar{U}, \bar{Z}) \wedge \bar{U} =_{\bar{U}_i} \bar{a}$.

We show the claim by contradiction. So assume that $\mathcal{C}_A$, $\mathcal{C}_{UPD}$, $\mathcal{C}_{DET}$ are false. We construct two databases D_1 and $D_2 = D_1 \cup \Delta$ that are both valid prior to the insertion $r(\bar{x}, \bar{a})$ (i.e., $\mathcal{Q}(D_1) = \mathcal{Q}(D_2) = V$, and both D_1 and D_2 satisfy the given FD's), but that derive different views after the insertion (i.e., $\mathcal{Q}(D_1 \cup r(\bar{x}, \bar{a})) \neq \mathcal{Q}(D_2 \cup r(\bar{x}, \bar{a}))$).

D_1 is taken to be the inverse database $\mathcal{Q}^{-1}(V)$ which is already known to be valid. Δ is constructed as follows:

1. Since $\mathcal{C}_A$, written as $\bigwedge_i (\exists \bar{D}_i)\varphi_i(\bar{D}_i)$ for shorthand, is false, there must be some i such that $(\exists \bar{D}_i)\varphi_i(\bar{D}_i)$ is false. Initially, for each such i, we include $S_i(\bar{a}_i, \bar{D}_i, \bar{N}_i)$ in Δ. In the rest of the construction, any mention of i will refer to these groups.

2. We use the FD's to chase $D_1 \cup \Delta$ until quiescence. Conflicts do not arise since both $\mathcal{C}_{UPD}$ and $\mathcal{C}_{DET}$ are false. After quiescence, $D_1 \cup \Delta$ essentially satisfies all the FD's.

3. During the chase, some variables in $\bar{D}_i$ (say $\bar{E}_i$) are bound (to say $\bar{e}_i$), some other (say $\bar{F}_i$) remain unbound. Consider the partition PART(S_i,$\bar{F}_i \cup \bar{N}_i$) into groups g_{ij}. Note that this partition is coarser than PART(S_i,$\bar{N}_i$) used in $\mathcal{C}_A$, but finer than PART(S,$\bar{D} \cup \bar{N}$). Let ψ_{ij} denote $(\exists \bar{U}, \bar{Z})[v(\bar{U}, \bar{Z}) \wedge \bar{U} =_{\bar{U}_{ij}} \bar{a} \wedge \bar{Z} =_{\bar{E}_{ij}} \bar{e}_i]$.

4. Remove from Δ all those $S_{ij}(\bar{a}_{ij}, \bar{e}_{ij}, \bar{F}_{ij}, \bar{N}_{ij})$ such that ψ_{ij} is true, and bind all remaining unbound variables (i.e. $\bar{F}_{ij}$ and $\bar{N}_{ij}$) to new constants.

We claim that $\mathcal{Q}(D_2 \cup r(\bar{x}, \bar{a})) \neq \mathcal{Q}(D_1 \cup r(\bar{x}, \bar{a}))$. It is easy to see that $\mathcal{Q}(D_2 \cup r(\bar{x}, \bar{a}))$ contains some tuple $(\bar{x}, \bar{a}, -)$, by construction of Δ. If at least a variable remains unbound when the chase reached quiescence, its value (a new constant) will show up in some tuple $(\bar{x}, \bar{a}, -)$ from $\mathcal{Q}(D_2 \cup r(\bar{x}, \bar{a}))$. This tuple $(\bar{x}, \bar{a}, -)$ cannot be in $\mathcal{Q}(D_1 \cup r(\bar{x}, \bar{a}))$ (because of the new constants). If all variables are bound by the chase, then necessarily both $\bar{F}_{ij}$ and $\bar{N}_{ij}$ are empty. Furthermore $\mathcal{Q}(D_1 \cup r(\bar{x}, \bar{a}))$ cannot have any $(\bar{x}, \bar{a}, -)$ since otherwise, any $S_{ij}(\bar{a}_{ij}, \bar{e}_{ij})$ would have hold in D_1, contradicting the hypothesis that some ψ_{ij} is false.

We now claim that $\mathcal{Q}(D_2) = V$. To prove our claim, we need to show that no tuples from Δ can contribute to the view (prior to insertion). Recall that only those groups g_{ij} such that ψ_{ij} is false do have tuples in Δ. If a Δ–tuple from a group g_{ij} contributes to the view, then all Δ–tuples from g_{ij} would also do so. Hence we can talk about group contribution instead of tuple contribution to the view. For each group g_{ij} such that ψ_{ij} is false, consider the set T_{ij} of D_1–tuples from S_{ij} that participate in the chase. Let $\{g_{ij} \mid j \in m_i\}$ be all the groups in the partition of S_i. Suppose the Δ–tuples from some of these groups that have ψ_{ij} false were to contribute to the view. Let n_i (a subset of m_i) denote these groups. Then D_1 must contain the following matching tuples:

- Tuples $S_{ik}(\bar{u}_{ik}, \bar{e}'_{ik}, \bar{f}_{ik}, \bar{n}_{ik})$ for all $k \in (m_i - n_i)$: $\bar{u}_{ik}$ agrees with $\bar{a}$ over $\bar{U}_{ij}$ for $j \in n_i$, $\bar{e}'_{ik}$ agrees with $\bar{e}_i$ over $\bar{E}_{ij}$ for all $j \in n_i$, all remaining constants in $\bar{u}_{ik}$ and $\bar{e}'_{ik}$ agree across groups over the same variables.

- Tuple $r(\bar{x}, \bar{u})$: $\bar{u}$ agrees with $\bar{a}$ over $\bar{U}_{ij}$ for all $j \in n_i$ and with all the $\bar{u}_{ik}$.

Since D_1 contains $r(\bar{x}, \bar{u})$, the view must have some tuple $v(\bar{x}, \bar{u}, \bar{z})$. The tuples $S_{ik}(\bar{u}_{ik}, \bar{e}'_{ik}, \bar{f}_{ik}, \bar{n}_{ik})$ for all $k \in (m_i - n_i)$, together with T_{ij} for all $j \in n_i$, should force $\bar{z}$ to agree with $\bar{e}_i$ over $\bar{E}_{ij}$ for all $j \in n_i$. Hence, ψ_{ij} must hold for all $j \in n_i$, which contradicts the fact that all groups g_{ij}, $i \in n_i$, have ψ_{ij} false. In conclusion, no tuples from Δ can contribute to the view.

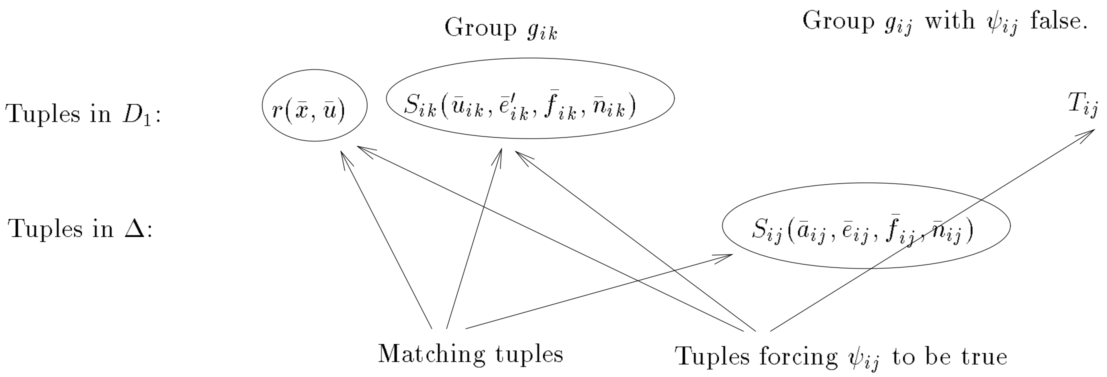


Figure 7: Δ cannot contribute to the view.