Efficient Maintenance of Materialized Mediated Views*

James Lu
Bucknell University
lu@sol.cs.bucknell.edu

Guido Moerkotte
Lehrstuhl für Informatik III
RWTH Aachen
moer@gom.informatik.rwth-aachen.de

Joachim Schü
Universität Karlsruhe
schue@ira.uka.de

V.S. Subrahmanian
University of Maryland
vs@cs.umd.edu

Abstract

Integrating data and knowledge from multiple heterogeneous sources — like databases, knowledge bases or specific software packages — is often required for answering certain queries. Recently, a powerful framework for defining mediated views spanning multiple knowledge bases by a set of constrained rules (cf. work of Kanellakis et. al. [27]) was proposed [39, 5, 26]. Within this paper, we investigate the materialization of these views by unfolding the view definition and the efficient maintenance of the resulting materialized mediated view in case of updates. Thereby, we consider two kinds of updates: updates to the view and updates to the underlying sources. For each of these two cases several efficient algorithms maintaining materialized mediated views are given. We improve on previous algorithms like the DRed algorithm [22] and introduce a new fixpoint operator W_P which — opposed to the standard fixpoint operator T_P [19] — allows us to correctly capture the update's semantics without any recomputation of the materialized view.

1 Introduction

Integrating data and knowledge from multiple heterogeneous sources each one possibly with a different underlying data model is not only an important aspect of automated reasoning but also of retrieval systems — in the widest sense — whose queries can span multiple such sources. These sources can be as different as relational or deductive databases, object bases, (constraint) knowledge bases, or even (structured) files and arbitrary program packages encapsulating specific knowledge, often in a hard-wired form accessible only through function calls. Many queries can only be answered if data and knowledge from these different sources are available. (For a

*Author for Correspondence: V.S. Subrahmanian, Dept. of Computer Science, Univ. of Maryland, College Park, MD 20742. Tel: (301) 405-2711; Fax: (301) 405-6707, Email: vs@cs.umd.edu. This research was supported by the Army Research Office under grant DAAL-03-92-G-0225, by the Air Force Office of Scientific Research under grant F49620-93-1-0065, by ARPA/Rome Labs contract Nr. F30602-93-C-0241 (Order Nr. A716), and by an NSF Young Investigator award IRI-93-57756. James Lu was supported by the NSF under grant number CCR9225037.

motivating example see Sec. 2.2.) In order to answer these queries, it is necessary to define a mediator [45, 46] integrating the different sources on a semantic level by providing an integrated view spanning these sources.

Traditional research on view or schema integration, and interoperability of databases concentrates on integrating databases, possibly with different underlying schemata or even data models [16, 29, 33, 36]. The basic idea often is to aim for a global integrating schema or view whose definition mediates between different databases. Only lately, investigations started to integrate other sources of data available. The most prominent example of such a source is the file. Lately, it was proposed to integrate (structured) files and object bases by providing an object base view on the file and a file view upon the object base [2, 20]. Again, though the means by which mediators are defined are different from the traditional ones, they form the basis for the integration. Another also recent but quite different application area for mediators are cooperative intelligent agents [10].

Another powerful technique — based on constraint logic — for integrating multiple knowledge bases is introduced in [39, 5, 26] and is based, in part, on the ideas of constrained databases due to Kanellakis, Kuper and Revesz [27]. While this work examines a framework for expressing mediated views, the paper [41] describes a concrete implementation of one such mediating system called HERMES (HEterogeneous Reasoning and MEdiator System). HERMES supports the integration of multiple databases and reasoning paradigms on both the PC/Windows and the SUN/Unix platforms and provides an environment which allows flexibility in adding new databases and software packages. In HERMES, mediators are expressed in a rule-based language containing a special predicate `in` used to achieve logical integration at the semantic level. It enables access to data contained in external databases, and gives HERMES the ability to execute functions in existing software – the current implementation of HERMES integrates PARADOX. INGRES, DBASE with third-party path planning packages, numerical computation packages, face recognition packages, and multimedia application packages.

As for traditional views, mediated views are materialized for efficiency reasons. A materialized view can be affected by two kinds of updates:

- updates to the materialized view, and
- updates to the underlying sources.

If an update of the first kind occurs to a view, whether materialized or not, the problem occurs on how to reflect the update correctly by changing the base tables appropriately. This problem is called the *view update problem* and has been discussed extensively for relational [6, 13, 15], deductive [6, 32, 42, 43], and object-oriented [1, 14, 31, 35] databases. However, our objective is slightly different. As motivated by an example in Section 2.2, we do not necessarily assume that an update occurring to a view has to be reflected within some underlying source. Instead, we assume that the view itself — or, to be more precise, its definition — is affected by the update. This kind of update affecting the view's definition is typically not treated within the view update literature. One exception are deductive databases, where the addition or deletion of rules to the definition of an intensional predicate is discussed [42]. However, they neither materialize nor preprocess the view for efficiency reasons.

Within the traditional context, the second case occurs if an update to a base table occurs which possibly affects a materialized view. The resulting problem — preserving the consistency of the view — is called *view maintenance* and has been discussed for, e.g., for (extended) relation

[8, 23, 38] and deductive databases [30, 24, 21, 44, 37]. The same problem occurs also for the materialization of functions within object bases [28]: if the values of some object's attributes change, the materialized function value becomes invalid. However, since we do not necessarily materialize the view upon the underlying sources of our mediated views but instead perform materialization by unfolding the view definition as independent as possible from the underlying sources, the traditional view maintenance problem occurs quite differently to us. Hence, the traditional view maintenance problem and our problem do not intersect but complement each other.

Subsequently, we treat both kinds of updates to materialized mediated views and show how they can be handled efficiently. More specifically, the primary aim is to specify how to efficiently maintain views of mediated systems such as those that may be constructed in HERMES when insertion and deletion requests of both of the above two kinds are made. As in the standard case, a *materialized view* in mediated systems may be thought of as a set of facts that can be concluded from the mediator rules. However, we show that more generally, a materialized mediated view may be regarded as a set of *constraint* atoms that are not necessarily ground. Taking materialized views to be sets of constrained atoms leads to a number of advantages:

1. First of all, it allows us to perform updates to *constrained databases* such as those described by Kanellakis et. al. [27]. To our knowledge, there are currently no methods to incrementally maintain views in constrained databases.
2. We show for updates of the second kind that even in the case of unconstrained databases, such as those considered by Gupta, Mumick and Subrahmanian [22], this approach leads to a simpler and more efficient deletion algorithm than the deletion algorithm, DRed presented in [22].
3. For updates of the first kind, we depart from using the standard fixpoint operation T_P as defined by Gabrielli and Levi [19]. Instead, we introduce the fixpoint operator W_P. W_P is able to capture updates of the second kind without any recomputation of the materialized mediated view while maintaining the semantics of T_P and correctly capturing the update.

The rest of the paper is organized as follows. Section 2.1 gives the preliminaries. Section 2.2 introduces the running example which also motivates the integration of multiple sources for answering a single query as well as the two kinds of updates. Section 2.3 formally defines the notion of materialized mediated view. Section 3 treats updates of the first kind whereas Section 4 treats updates of the second kind. Section 5 discusses related work and Section 6 concludes the paper.

2 Preliminaries and Motivating Example

2.1 Syntax and Semantics

In this section, we will briefly describe the basic theory behind mediated systems proposed in [26, 39, 4, 5, 3]. Illustration is provided via the HERMES implementation.

A *domain*, $\mathcal{D}$, is an abstraction of databases and software packages and consists of three components: (1) a set, Σ whose elements may be thought of as the data-objects that are being manipulated by the package in question, (2) a set $\mathcal{F}$ of functions on Σ – these functions take objects in Σ as input, and return, as output, objects from their range (which needs to be specified). The functions in $\mathcal{F}$ may be thought of as the predefined functions that have been implemented in the software package being considered, (3) a set of relations on the data-objects in Σ – intuitively, these relations may be thought of as the predefined relations in the domain, $\mathcal{D}$.

A constraint Ξ over D is a first order formula where the symbols are interpreted over D. Ξ is either true or false in D, in which case we say that Ξ is solvable, or respectively unsolvable in D, where the reference to D will be eliminated if it is clear from context. The key idea behind a mediated system is that constraints provide the link to external sources, whether they be databases, object bases, or other knowledge sources. This idea is developed extensively in [26, 39, 4, 5, 3] and we do not elaborate on them here.

For example in HERMES, a *domain call* is a syntactic expression of the form

$$\texttt{domainname} : \langle\texttt{domainfunction}\rangle(\langle\texttt{arg1}, \ldots, \texttt{argn}\rangle)$$

where `domainfunction` is the name of the function, and $\langle\texttt{arg1}, \ldots, \texttt{argn}\rangle$ are the arguments it takes. Intuitively, a domain call may be read as: in the domain called `domainname`, execute the function `domainfunction` defined therein on the arguments $(\texttt{arg1}, \ldots, \texttt{argn})$. The *result* of executing this domain call is coerced into a set of entities that have the same type as the output type of the function `domainfunction` on the arguments $(\texttt{arg1}, \ldots, \texttt{argn})$

A *domain-call atom* (DCA-atom) is of the form $\texttt{in}(\texttt{X}, \texttt{domainfunction}\rangle(\langle\texttt{arg1}, \ldots, \texttt{argn}\rangle)$ where `in` is a constraint that is satisfied just in case the entity `X` is in the set returned by the domain call in the second argument of `in(-,-)`. In other words, `in` is the polymorphic set membership predicate.

More concretely, `in(A,paradox:select_eq('phonebook',"name","john smith"))` is a DCA-atom that is true just in case `A` is a tuple in the result of executing a selection operation (finding tuples where the `name` field is `john smith`) on a relation called `phonebook` maintained in a `PARADOX` database system.

A *mediator/constrained database* is a set of rules of the form

$$A \quad \leftarrow \quad D_1 \wedge \ldots \wedge D_m || A_1, \ldots, A_n.$$

where $A, A_1, \ldots, A_n$ are atoms, and $D_1, \ldots, D_m$ are DCA-atoms. Note that for simplicity, we restrict constraints to DCA-atoms of the form described above. This does not however, detract from the generality of the techniques described in the paper – It can be shown (Example 2 presents one example of how to do this) that all the kinds of constraints considered by Kanellakis et. al. can be captured within this framework (Lu, Nerode, Subrahmanian present further details [26]).

2.2 Motivating Example

We introduce a running example which also motivates our approach. This example has been addressed in the existing HERMES implementation [40].

Example 1 (Law-Enforcement Example) Consider the problem of identifying all people P who have been recorded, by surveillance cameras, as having met with an individual X (for instance, X may be a Mafia chief like Don Corleone), who live within a hundred mile radius of Washington DC, and who work for a suspected front company "ABC Corp." Solving this problem may require access to a wide variety of data structures, databases, and furthermore, require recourse to diverse *reasoning paradigms* as well. For instance:

- First, we need to access a background **face database** containing pictures (e.g. passport pictures) of individuals. In this face database, the identity of the photographed individuals is known.
- Second, we need access to a database containing **surveillance photographs**. These photographs may have been obtained by using surveillance cameras.
- Third, we need access to **face-extraction algorithms** that extract the "prominent" faces from the images generated by the surveillance camera.
- Fourth, we need methods of **matching** faces extracted from the surveillance data by the face-extraction algorithm, so as to be able to figure out who appears in which images.
- Fifth, we may need to access a **relational database** (e.g. a phone/address book database) specifying the names, addresses, and phone numbers of individuals. This database may be stored as a relation in a well known relational DBMS, say PARADOX.
- Sixth, we may need to access a **spatial database/spatial reasoning package** in order to determine whether a given address lies within 100 miles of Washington DC.
- Finally, we may need to access a **second relational database** about the employees of ABC Corp. Note that this relational database maybe completely different from the phone/address book relational database alluded to earlier in this example, and may be stored as a DBASE relation.

In order to answer the above query, we must be able to integrate the above software packages at the software level, as well as at the logical level. In this paper, we will not go into the software integration scheme – it is described in [41], but we will go into some details about the mediator syntax itself in order to define what "soundness and completeness" of view maintenance means, and in order to develop algorithms for view maintenance that are sound and complete. For this example, the mediator may be expressed as three clauses:

$$
\begin{aligned}
\mathtt{seenwith(X,Y)} \leftarrow\ & \mathtt{in(P1, facextract : segmentface('surveillancedata'))} \wedge \\
& \mathtt{in(P2, facextract : segmentface('surveillancedata'))} \wedge \\
& = \mathtt{(P1.origin, P2.origin)} \wedge \\
& \mathtt{P1} \neq \mathtt{P2} \wedge \\
& \mathtt{in(P3, facedb : findface(X))} \wedge \\
& \mathtt{in(true, facextract : matchface(P1, P3))} \wedge \\
& \mathtt{in(Y, facedb : findname(P3))}. \qquad (1) \\
\mathtt{swlndc(X,Y)} \leftarrow\ & \mathtt{seenwith(X,Y)} \wedge \\
& \mathtt{in(A, paradox : select\_eq('phonebook', "name", X))} \wedge \\
& \mathtt{in(Pt1, spatialdb : locateaddress(A.streetnum, A.streetname, A.cityname)} \wedge \\
& \mathtt{A.statename, A.zipcode))} \wedge \\
& \mathtt{in(true, spatialdb : range('dcareamap', Pt1.X, Pt2.X, 100))}. \qquad (2) \\
\mathtt{suspect(X,Y)} \leftarrow\ & \mathtt{swlndc(X,Y)} \wedge \\
& \mathtt{in(Tuple, dbase : select\_eq('empl\_abc', "name", Y))}. \qquad (3)
\end{aligned}
$$

The `seenwith` predicate access a domain called `faceextract` which is a pattern recognition package that uses a function called `segmentface` to locate the faces in a set of photographs, and then extracts these faces (leading to "mugshots") which are then stored in files. The extraction procedure returns a list of pairs of the form ($<$ `resultfile`, `origin` $>$) specifying which image in the surveillance data, a given face was extracted from (the origin) and where the mugshot/face is now stored. The `faceextract` domain also contains a function called `matchface` that takes as face (such as those extracted by the `faceextract` domain) and checks if this face is identical to another face in the mugshot library. Likewise, the `seenwith` predicate access a domain called `facedb` containing a function called `findface` which determines, given a person's name, whether his face is in a mugshot library. The `facedb` domain also contains a function called `findname` which, given a mugshot in the mugshot library, returns the name of the person involved.

Given that a person `Y` has been `seenwith X`, `swlndc` (for "seen with and lives near DC"), accesses a relational database to find the address of `Y`, and then accesses a spatial data management system to determine what $(\mathtt{x}, \mathtt{y})$ coordinates, on a map of the DC area, this address corresponds to (using a function called `locateaddress`). It then determines, using a function called `range`, whether this address lies within the specified distance from DC.

Finally, a person `Y` is a suspect just in case `swlndc("Don Corleone", Y)` is true and if he is an employee of "ABC Corp." For this, a DBASE relation called `empl_abc` is accessed. The above three clauses express the mediator for this example in its entirety. Figure 1 shows the logical arrangement of the domains being integrated.

Example 2 (Constrained Databases) Kanellakis et. al. [27] have introduced the concept of constrained databases, which can be modeled within our framework (a formal proof is contained in [26]). For instance, if we wish to write constraints over the `arith`metic domain, then we may have *functions* called `great(X)` that returns as output, the set of all integers greater than `X`. (Note that when implementing this, the entire — infinite — set need not be computed all at once — parts of may be computed and stored in a buffer with additional computation being performed on a need to perform basis. How to do this is not the topic of this paper, see [26] for details). Likewise, `plus(X, Y)` returns the *singleton set* $\{\mathtt{X} + \mathtt{Y}\}$.

In the rest of this paper, we will use these examples to motivate various kinds of updates that may occur and that bear an important relationship to view maintenance in such mediated systems.

2.3 Non-Ground Materialized Mediated Views

In this section, we will define the concept of a materialized mediated view. Typically, a materialized view is a set of ground atoms, corresponding to a set of relations whose fields are filled in with (ground) values. In our case, a materialized view will generalize this notion, allowing non-ground atoms to occur in it, as long as the variables in the atom satisfy certain constraints which are defined as follows:

- Any DCA-atom is a constraint.
- If `X` is a variable symbol and `T` is either a variable symbol, or a constant, then $\mathtt{X} = \mathtt{T}$ and $\mathtt{X} \neq \mathtt{T}$ are constraints.
- Any conjunction of constraints is a constraint.

Thus, for example, $\mathtt{X} = \mathtt{2} \wedge \mathtt{Y} \neq \mathtt{X} \wedge \mathtt{in}(\mathtt{Y}, \mathtt{arith}: \mathtt{greater}(\mathtt{X}))$ is a constraint in the domain `arith` described earlier. A more common way of writing this constraint is $\mathtt{X} = \mathtt{2} \wedge \mathtt{Y} \neq \mathtt{X} \wedge \mathtt{Y} \geq \mathtt{X}$ – we will use this notation when referring to the numeric domain.

A *constrained atom* is an expression of the form $A(\vec{X}) \leftarrow \Xi$ where $\vec{X}$ denotes a tuple of variables and Ξ is a constraint.

Given a constrained atom $A(\vec{X}) \leftarrow \Xi$, $[A(\vec{X}) \leftarrow \Xi]$ denotes the set of instances of X that are solutions of Ξ, viz. $\{A(\vec{X})\theta \mid \theta \text{ is an solution of } \Xi\ \}$. For example, taking the same constraint $\Xi = (\mathtt{X} = \mathtt{2} \wedge \mathtt{Y} \neq \mathtt{X} \wedge \mathtt{Y} \geq \mathtt{X})$ as above, $[\mathtt{p}(\mathtt{X}, \mathtt{Y}) \leftarrow \Xi]$ is the set $\{\mathtt{p}(2,3), \mathtt{p}(2,4), \mathtt{p}(2,5), \ldots\}$. If $\mathcal{C}$ is a set of constrained atoms, $[\mathcal{C}]$ is defined to be $\bigcup_{A(X) \leftarrow \Xi \in \mathcal{C}} [A(X) \leftarrow \Xi]$.

An interpretation for a mediated system P is any set of constrained atoms. A constrained atom $A(\vec{X}) \leftarrow \Xi$ is said to be *true in an interpretation* I iff $[A(\vec{X}) \leftarrow \Xi] \subseteq [I]$. Given a constrained database P we may define an operator, T_P that maps interpretations to interpretations in the following way:

$$\begin{aligned} T_P(I) \quad = \quad & \{A(\vec{X}) \leftarrow \Xi \mid \\ & \text{There is a clause } A(t_0) \leftarrow \Xi_0 || A_1(t_1), \ldots, A_n(t_n) \text{ in P} \\ & \forall 1 \leq i \leq n \ : \ \exists A_i(X_i) \leftarrow \Xi_i \in I, \\ & \text{which share no variables and the constraint} \\ & \Xi = \Xi_0 \wedge \Xi_1 \wedge \ldots \wedge \Xi_n \wedge \{\vec{X}_1 = \vec{t}_1\} \wedge \ldots \wedge \{\vec{X}_n = \vec{t}_n\} \wedge \{\vec{X} = \vec{t}_0\} \text{ is solvable}\} \end{aligned}$$

Note that each $\vec{t}_i$ is assumed to be a tuple of terms of the same length as X_i. This operator was originally defined by Gabbrielli and Levi [19] who used it to define a non-ground representation of the ground least Herbrand model of a constrained database/logic program. For the types of updates that we consider in Sections 3 and 4, this non-ground set of constrained atoms constitutes the materialized view of the constrained database which we are interested in maintaining. The iteration of T_P is defined in the usual way.

An important point to note is that T_P may often yield a set containing multiple atoms of the form $A(\vec{X}) \leftarrow \Xi_1; \ldots; A(\vec{X}) \leftarrow \Xi_m$ where the constraints, $\Xi_1, \ldots, \Xi_m$ are not necessarily incompatible. This corresponds to an extension, to the case of constrained databases, of the well-known *duplicate semantics* proposed by Mumick [34] in the context of ordinary deductive databases.

3 Updating Views

In our context, view updating deals with the following problem: given a constrained database P, a materialized view MMV, and an update u, compute a new materialized view that accurately reflects this update. Note that we adapt the view and not modify the underlying sources. Remember that a materialized view is a set of constrained atoms. An *update* may take one of the following three forms:

- **Atom Addition:** A constrained atom (involving predicates defined in the mediator) is added to the materialized view. For instance, in the Law Enforcement example, the atom `seenwith(“Don Corleone”, “Jane Doe”)` may be inserted into the materialized view (even though this fact may not be derivable using clause (1) of the paper.) This may be due to the fact that some external reasons (e.g. a policeman saw them together and duly reported it) may justify its truth.

- **Atom Deletion:** Suppose the atom `suspect("Don Corleone", "John Smith")` was in the materialized view (e.g. it may been derivable from the original constrained database), but we may wish to delete this fact because there is external evidence that John Smith has no connection with Don Corleone (e.g. he may have been derived as a suspect because he was in a large crowd of people one of whom was Don Corleone).

- **External Data Changes and Function Modification:** In a mediated system, the mediator accesses (potentially) many different databases and/or data structures. The data contained in those databases/data structures may be updated, triggering changes to the data in the materialized view. For instance, in the Law Enforcement Example, it may turn out that the surveillance data has been extended (through the addition of new photographs, say) and hence, the domain call `facextract:segmentface ('surveillancedata')` returns a set of objects that are different from what was returned by this function prior to the update. This change in the domain is modeled as a change in the *function* which, in this example, happens to be `segmentface`. Changes of this kind may trigger new changes to the materialized view (for instance, adding new pictures will, presumably, enlarge the pool of suspects). We will show how this intuition of modeling changes in local databases as function updates leads to simple algorithms for updating a mediated materialized view.

Note that we do not consider the problem of adding or deleting a rule from the mediator.

3.1 Deletion of Constrained Atoms

In this section, we will present two algorithms that will compute a materialized view obtained by deleting an existing atom from the mediated materialized view. The correctness of these algorithms is shown and the relative areas of applicability are discussed. In particular, we emphasize that both algorithms apply to non-recursive, as well as recursive views (Example 6 provides an example).

Declarative Semantics of Constrained-Atom Deletion: Let $A(\vec{X}) \leftarrow \Theta$ be a constrained atom whose instances are to be deleted from the materialized view M. Let Del be the set $\{A(\vec{Y}) \leftarrow \Theta \wedge (\vec{X} = \vec{Y}) \wedge \Lambda \mid$ where $A(\vec{Y}) \leftarrow \Lambda$ is a constrained atom in the materialized view, MMV and $\Xi = \Theta \wedge (\vec{X} = \vec{Y}) \wedge \Lambda$ is satisfiable$\}$. Del is the initial input to our deletion algorithm below. Observe that the construction of Del ensures that only those constrained atoms that are actually in the existing materialized view will be deleted. We now show how to construct a new constrained database P' which accomplishes the deletion of these atoms as well as the deletion of their consequences. The least model of this constrained database will be the desired materialized view *after* the deletions are performed. Hence, P' provides the declarative semantics of the deletion operation, and we will later show in Algorithm 1, how this declarative semantics can be computed.

$\boxed{\text{Rewrite the Constrained Database } P}$ resulting in a *new* constrained database P', as follows.

1. If $A(\vec{X}) \leftarrow \Xi \mid\mid B_1(\vec{X}_1), \ldots, B_n(\vec{X}_n)$ is in P and $A(\vec{Y}) \leftarrow \Xi'$ is in Del, then
$$A(\vec{X}) \leftarrow \Xi \wedge \mathbf{not}(\Xi') \wedge (\vec{X} = \vec{Y}) \mid\mid B_1(\vec{X}_1), \ldots, B_n(\vec{X}_n) \text{ is in } P'. \tag{4}$$

2. Any clause in P with a head different from $A(\vec{X})$ is in P'.

We present two algorithms for accomplishing the above deletion. The first algorithm extends the DRed algorithm of Gupta, Mumick and Subrahmanian [22] to the mediated case. It is efficient when the mediated view is duplicate-free, i.e. when, for all distinct constrained atoms $A(\vec{X}) \leftarrow \Xi_1$ and $A(\vec{Y}) \leftarrow \Xi_2$ in the materialized view, $[A(\vec{X}) \leftarrow \Xi_1] \cap [A(\vec{Y}) \leftarrow \Xi_2] = \emptyset$. The second algorithm shows how to completely eliminate the expensive rederivation step in this algorithm, thus improving the DRed algorithm. Furthermore, the second algorithm uses the least fixpoint of the Gabbrielli-Levi operator with no changes (in particular, duplicate checking and elimination, required in Algorithm 1, are not required either).

3.1.1 The First Deletion Algorithm

Algorithm 1 (Extended DRed Algorithm)

1. Unfold the constrained atoms to be deleted with respect to the *original* constrained database P, so as to compute a set of constraint base facts, that are to be "possibly deleted".

$$
\begin{array}{rcl}
P\_OUT_0 & = & Del \\
P\_OUT_{k+1} & = & \{B(\vec{X}) \leftarrow \Xi \mid \\
& & \text{There is a clause } B(\vec{X}) \leftarrow \Xi_0 \mid\mid B_1(\vec{X}'_1), \ldots, B_n(\vec{X}'_n) \text{ in } P \\
& & \text{and for at least one} j \in \{1, \ldots, n\} : B_j(\vec{X}_j) \leftarrow \Xi_j \in P\_OUT_k \\
& & \forall i \neq j \in \{1, \ldots, n\} : B_i(\vec{X}_i) \leftarrow \Xi_i \text{ is a constraint} \\
& & \text{atom from the materialized view } M = T_P \uparrow \omega(\emptyset), \\
& & \text{and} \Xi = \Xi_0 \wedge \ldots \wedge \Xi_n \wedge \{\vec{X}_1 = \vec{X}'_1\} \wedge \ldots \wedge \{\vec{X}_n = \vec{X}'_n\} \\
& & \text{is satisfiable}\} \\
P\_OUT & = & \bigcup_{k \geq 0} P\_OUT_k
\end{array}
$$

   Note that the members of P_OUT are candidates for deletion from the materialized view, but not all of them will necessarily be deleted.

2. Compute an overestimate, M', of necessary deletions with $[M'] = [M] \backslash [P\_OUT]$ as follows:

   (a) For every $B(X_1) \leftarrow \Xi$ in M for which there exists a $B(\vec{X}_2) \leftarrow \Gamma$ in P_OUT,

   $$B(\vec{X}_2) \leftarrow \mathbf{not}(\Gamma) \wedge \Xi \wedge (\vec{X}_1 = \vec{X}_2) \text{ is in } M' \tag{5}$$

   (b) For each remaining constraint fact $B(\vec{X}) \leftarrow \Xi$ in M, $B(\vec{X}) \leftarrow \Xi$ is in M'.

3. Rederive the new view by computing $T_{P''} \uparrow \omega(M')$. **Return this as output.**
   P'' is obtained from P' by considering each clause $C \equiv A(\vec{X}) \leftarrow \Xi || B_1 \& \ldots \& B_n$ in P' as follows:

   (a) if $A \leftarrow \Xi_1$ is true in M', then delete C from P'.

   (b) Otherwise, eliminate all B_i's from the body of this clause that are true in M'.

(c) If all rules involving a predicate A have been eliminated by Step 3a, then eliminate all clauses with that predicate in the body. This process should be repeated until no more rules can be eliminated.

The reason for the incrementality of the above algorithm is that Step 3 eliminates a large part of the constrained database from consideration by either eliminating rules, or eliminating various preconditions in the bodies of rules.

Theorem 1 *Let $X = T_{P''} \uparrow \omega(M')$ be the output of Algorithm 1. Then: $[X] = [T_{P'} \uparrow \omega(\emptyset)]$, i.e. the algorithm is correct.*

Note that there are multiple ways of representing equivalent constraint atoms (e.g. $p(X,Y) \leftarrow X = Y + 1$ and $p(X,Y) \leftarrow Y = X - 1$ are syntactically different, but semantically equivalent). The above result says that the set of solutions of the constraint atoms returned by the algorithm coincide with the intended declarative semantics.

Example 3 Suppose the materialized mediated view associated with the Law Enforcement example contains the atoms: `seenwith`(“`Don Corleone`”, “`John`”), `seenwith`(“`Don Corleone`”, “`Ed`”); `swlndc`(“`Don Corleone`”, “`John`”); `swlndc`(“`Don Corleone`”, “`Ed`”). Suppose we are interested in deleting `seenwith`(“`Don Corleone`”, “`John`”); this may be due to external information (e.g. that the photograph was a forgery intended to frame John) then the materialized view will be updated by the deletion of the two atoms

`seenwith`(“`Don Corleone`”, “`John`”) and ; `swlndc`(“`Don Corleone`”, “`John`”).

These two atoms constitute the set P_OUT. In this example, all atoms in P_OUT are in fact deleted.

Example 4 Suppose we consider the *constrained database* containing: $\{A(X) \leftarrow X \leq 3; A(X) \leftarrow B(X); B(X) \leftarrow X \leq 5; C(X) \leftarrow A(X)\}$; the materialized view associated with this is: $\{A(X) \leftarrow X \leq 3; A(X) \leftarrow X \leq 5; B(X) \leftarrow X \leq 5; C(X) \leftarrow X \leq 3; C(X) \leftarrow X \leq 5\}$. Suppose we wish to delete $B(X) \leftarrow X = 6$. Then the set $Del = \{B(X) \leftarrow X = 6\}$. Then P_OUT contains: $\{B(X) \leftarrow X = 3; A(X) \leftarrow X = 3; C(X) \leftarrow X = 3\}$ (actually in this example, we are showing a simplified version of the constraints). Note that in this case, $A(X) \leftarrow X = 3$ and $C(X) \leftarrow X = 3$ should *not* be eliminated from the view because $A(X) \leftarrow X = 3$ has a proof independently of the proof that depends upon $B(X) \leftarrow X = 3$. M', as presented in the Extended DRed algorithm now becomes $\{A(X) \leftarrow X \leq 3 \wedge X \neq 3; A(X) \leftarrow X \leq 5 \wedge X \neq 3; B(X) \leftarrow X \leq 5 \wedge X \neq 3; C(X) \leftarrow X \leq 3 \wedge X \neq 3; C(X) \leftarrow X \leq 5 \wedge X \neq 3;\}$. The constrained database P' used in the definition of the extended DRed algorithm is identical to P except that $B(X) \leftarrow X \leq 5$ is replaced by $B(X) \leftarrow X \leq 5 \wedge X \neq 3$; P'' is then the constrained database that contains just the rules $A(X) \leftarrow X \leq 3$ and $C(X) \leftarrow A(X)$. $T_{P''} \uparrow \omega(M')$ quickly evaluates to the materialized view, $A(X) \leftarrow X \leq 3; A(X) \leftarrow X \leq 5; B(X) \leftarrow X \leq 5 \wedge X \neq 3; C(X) \leftarrow X \leq 3; C(X) \leftarrow X \neq 3 \wedge X \leq 5\}$, which is the correct, final materialized view.

3.1.2 The Second Deletion Algorithm

We now present a second algorithm to accomplish the deletion of constrained atoms from materialized mediated views in which duplicates are retained. The important advantage of the new algorithm is the elimination of the rederivation step (Step 3) of the first algorithm. To achieve this, we assume that each constraint atom in the materialized view is "indexed" by a sequence of clauses representing the derivation of the constraint atom in T_P. For simplicity we may assume that clauses are numbered in the constrained database and we use $Cn(C)$ to denote the clause number of the clause C.

For each constraint atom $A(\vec{X}) \leftarrow \Xi$ in the materialized view $T_P \uparrow \omega(\emptyset)$, we associate an "index" sequence, called the *support* of $A(\vec{X}) \leftarrow \Xi$ and denoted $spt(A(\vec{X}) \leftarrow \Xi)$, as follows:

1. If $A(\vec{X}) \leftarrow \Xi \in T_P \uparrow 0$, then $spt(A(\vec{X}) \leftarrow \Xi) = \langle Cn(C) \rangle$ where C is the clause from which $A(\vec{X}) \leftarrow \Xi$ is derived in T_P.
2. Suppose $A(\vec{X}) \leftarrow \Xi \in T_P \uparrow n$. By definition there is a clause $C \in P$ of the form
$$A(\vec{Y}) \leftarrow \Xi_0 || B_1(\vec{X}_1), ..., B_k(\vec{X}_k)$$
such that $B_i(\vec{Y}_i) \leftarrow \Xi_i \in T_P \uparrow (n-1)$ and $\Xi = \Xi_0 \wedge_{i=1}^k \Xi_i \wedge (\vec{X} = \vec{Y}) \wedge_{i=1}^k (\vec{X}_i = \vec{Y}_i)$ is solvable. Then $spt(A(\vec{X}) \leftarrow \Xi) = \langle Cn(C), spt(B_1(\vec{Y}_1) \leftarrow \Xi_1), \ldots, spt(B_k(\vec{Y}_k) \leftarrow \Xi_k) \rangle$.

Observe that the support of any constraint atom is always finite. Moreover, each constraint atom in $T_P \uparrow \omega(\emptyset)$ possesses a unique support.

Lemma 1 Suppose $spt(F_1) = spt(F_2)$. Then F_1 and F_2 are the same constraint atom in $T_P \uparrow \omega(\emptyset)$.

The input to the algorithm is the same set Del given to Algorithm 1. The intuitive idea behind the algorithm is that the support of a constraint atom F is used for determining whether an earlier deletion affects the deletion of F. We present the algorithm first followed by several examples.

Algorithm 2 (The Straight Delete (StDel) Algorithm)

1. Let M be the materialized view given by $T_P \uparrow \omega(\emptyset)$ and mark each constraint atom in M.
2. For each constraint atom $F = A(\vec{X}) \leftarrow \Xi$ in M where there exists $A(\vec{Y}) \leftarrow \Gamma \in Del$, such that $\Xi \wedge (\vec{X} = \vec{Y}) \wedge \Gamma$ is solvable, replace F with the new constraint atom
$$A(\vec{X}) \leftarrow \Xi \wedge (\vec{X} = \vec{Y}) \wedge not(\Gamma).$$
In addition, put the pair $(A(\vec{Y}) \leftarrow \Xi \wedge (\vec{X} = \vec{Y}) \wedge \Gamma, spt(F))$ into P_OUT.
3. **repeat**
For each constraint atom $F = A(\vec{X}) \leftarrow \Xi$ in M that is marked. Suppose $spt(F) = \langle Cn(C), s_1, ..., s_n \rangle$ for some constrained clause C having the form
$$A(\vec{Y}) \leftarrow \Xi_0 \| B_1(\vec{t}_1), ..., B_j(\vec{t}_j), ...B(\vec{t})_m,$$
and
   (a) The constraint atom $(B_j(\vec{Y}_j) \leftarrow \Xi_j, s_j)$, for some $1 \leq j \leq n$, is in P_OUT.

(b) For each $1 \leq i \leq n$ such that $i \neq j$, the constraint fact $F_i = B_i(\vec{Y_i}) \leftarrow \Xi_i$ with $s_i = spt(F_i)$ is in M.

(c) The constraint $\Xi_0 \wedge \Xi \wedge (\vec{X} = \vec{Y}) \wedge \wedge_{i=1}^{n}(\vec{Y_i} = \vec{t_i} \wedge \Xi_i)$ is solvable.

Then replace F with

$$A(\vec{X}) \leftarrow \Xi_0 \wedge \Xi \wedge (\vec{X} = \vec{Y}) \wedge \wedge_{i=1}^{n}(\vec{Y_i} = \vec{t_i}) \wedge \Xi_1 \wedge ... \wedge not(\Xi_j) \wedge \ldots \wedge \Xi_n.$$

In addition, put the pair $(A(\vec{X}) \leftarrow \Xi \wedge (\vec{X} = \vec{Y}) \wedge \wedge_{i=1}^{n}(\vec{Y_i} = \vec{t_i} \wedge \Xi_i), spt(F))$ into P_OUT.

Until no remaining marked elements can be replaced.

4. Remove any constraint atom from M whose constraint is not solvable.

Note that the constraints that are created in step 3 of the algorithm will often contain redundancy. But as the next example illustrates, in many cases the redundancy can be removed by simplification of the constraints.

Example 5 Suppose P is the constrained database:

1. $A(X) \leftarrow X \leq 3$
2. $A(X) \leftarrow B(X)$
3. $B(X) \leftarrow X \geq 5$
4. $C(X) \leftarrow A(X)$

The materialized view of P is shown below on the left, where the corresponding support for each constraint atom is shown to the right.

$A(X) \leftarrow X \leq 3$	$\langle 1 \rangle$
$A(X) \leftarrow X \geq 5$	$\langle 2, \langle 3 \rangle \rangle$
$B(X) \leftarrow X \geq 5$	$\langle 3 \rangle$
$C(X) \leftarrow X \leq 3$	$\langle 4, \langle 1 \rangle \rangle$
$C(X) \leftarrow X \geq 5$	$\langle 4, \langle 2, \langle 3 \rangle \rangle \rangle$

Suppose the constraint atom $B(X) \leftarrow X = 6$ is specified for deletion. The declarative semantics of this deletion is given by the least fixpoint of the constrained database P':

$A(X) \leftarrow X \leq 3$
$A(X) \leftarrow B(X)$
$B(X) \leftarrow X \geq 5 \wedge X \neq 6$
$C(X) \leftarrow A(X)$

The corresponding materialized view $T_{P'} \uparrow \omega(\emptyset)$ contains the constraint atoms:

$A(X) \leftarrow X \leq 3$
$A(X) \leftarrow X \geq 5 \wedge X \neq 6$
$B(X) \leftarrow X \geq 5 \wedge X \neq 6$
$C(X) \leftarrow X \leq 3$
$C(X) \leftarrow X \geq 5 \wedge X \neq 6.$

The StDel algorithm achieves the equivalent view working as follows. Initially, each of the five constraint atoms in M is marked. In the second step, we replace $B(X) \leftarrow X \geq 5$ by the new constraint atom $B(X) \leftarrow X \geq 5, X \neq 6$, and put $(B(X) \leftarrow X \geq 5 \wedge X = 6, \langle 3 \rangle)$ into P_OUT where $\langle 3 \rangle$ is the support of the replaced constraint atom.

Next according to step 3 of the algorithm, we search for marked constraint atoms in M whose support contains $\langle 3 \rangle$. The only constraint atom that satisfies this condition is $A(X) \leftarrow X \geq 5$, whose support is $\langle 2, \langle 3 \rangle \rangle$. We construct from constrained clause 2 the new constraint atom $A(X) \leftarrow (X \geq 5) \wedge not(X \geq 5 \wedge X = 6)$ that replaces $A(X) \leftarrow X \geq 5$. Simplification of the constraint yields $A(X) \leftarrow X \geq 5 \wedge X \neq 6$. The pair $(A(X) \leftarrow X \geq 5 \wedge X = 6, \langle 2, \langle 3 \rangle \rangle)$ is then placed in P_OUT.

The next iteration of the algorithm finds that the support for the marked constraint atom $C(X) \leftarrow X \geq 5$ contains the support $\langle 2, \langle 3 \rangle \rangle$. Hence by a similar analysis as the previous paragraph, a replacement of this constraint atom by $C(X) \leftarrow X \geq 5 \wedge X = 6$ is made. The pair $(C(X) \leftarrow X \geq 5 \wedge X = 6, \langle 4, \langle 2, \langle 3 \rangle \rangle \rangle)$ is put into P_OUT.

The final iteration of step 3 does not produce any new replacement since the only remaining marked constraint atoms are $A(X) \leftarrow X \leq 3$ and $C(X) \leftarrow X \leq 3$. Neither of which possesses a support that contains a sub-support in P_OUT. Hence the algorithm terminates. □

Several observations are in order here. First, the supports that we use are similar to *justifications* used in truth-maintenance systems[17] in that they provide a "history" of the derivation of constraint atoms. However, they serve different purposes. The main difference between truth maintenance systems (TMSs) and view maintenance systems (VMSs) is that in TMSs, we attempt to delete an atom A by making it unprovable; in contrast, in view maintenance, we try to determine what atoms need to be deleted based on deleting A. For instance, let $P = \{a \leftarrow b; a \leftarrow c; b; c; d \leftarrow a\}$ and suppose we wish to delete a from the original materialized view $\{a, b, c, d\}$. Then view maintenance simply says that the new materialized view is $\{b, d\}$; in contrast, TMSs would find three "extensions" for this problem based on different ways of eliminating the derivability of a; these extensions lead to the *multiple* materialized views $\{b, c\}$ obtained by eliminating the first two formulas in P; $\{b\}$ obtained by eliminating the first and fourth formulas in P; and $\{b\}$ again obtained by eliminating the second and fourth formulas in P. This strategy is unsuitable for databases because we would like view maintenance to lead to a *single* resulting materialized view, not a possibly exponential number of materialized views such as may be the case with TMSs.

Secondly, the algorithm differs from the counting algorithm of [21] since here, each constraint atom in the materialized view corresponds to a single proof. The counting algorithm maintains a count of the number of proofs of an atom, but does not distinguish between different derivations. In contrast, in our algorithm, given any constrained atom $A(X) \leftarrow \Xi$, we maintain a *list* of supports.

Example 6 (Recursive Views) Suppose we consider the constrained database:

1. $P(X, Y) \leftarrow X = a \wedge Y = b$
2. $P(X, Y) \leftarrow X = a \wedge Y = c$
3. $P(X, Y) \leftarrow X = c \wedge Y = d$
4. $A(X, Y) \leftarrow P(X, Y)$
5. $A(X, Y) \leftarrow P(X, Z), A(Z, Y)$

The materialized view M is displayed below.

1. $P(X,Y) \leftarrow X = a \wedge Y = b$ $\quad\langle 1\rangle$
2. $P(X,Y) \leftarrow X = a \wedge Y = c$ $\quad\langle 2\rangle$
3. $P(X,Y) \leftarrow X = c \wedge Y = d$ $\quad\langle 3\rangle$
4. $A(X,Y) \leftarrow X = a \wedge Y = b$ $\quad\langle 4,\langle 1\rangle\rangle$
5. $A(X,Y) \leftarrow X = a \wedge Y = c$ $\quad\langle 4,\langle 2\rangle\rangle$
6. $A(X,Y) \leftarrow X = c \wedge Y = d$ $\quad\langle 4,\langle 3\rangle\rangle$
7. $A(X,Y) \leftarrow X = X' \wedge Z = Y' \wedge X' = a \wedge Y' = c \wedge Z = X'' \wedge$
   $Y = Y'' \wedge X'' = c \wedge Y'' = d$ $\quad\langle 4,\langle 2\rangle,\langle 4,\langle 3\rangle\rangle\rangle$

Suppose $Del = \{P(X,Y) \leftarrow X = c \wedge Y = d\}$. The view of the modified program P', when materialized, yields the set M'

$$
\begin{array}{lll}
P(X,Y) \leftarrow & X = a \wedge Y = b & \langle 1\rangle \\
P(X,Y) \leftarrow & X = a \wedge Y = c & \langle 2\rangle \\
A(X,Y) \leftarrow & X = a \wedge Y = b & \langle 4,\langle 1\rangle\rangle \\
A(X,Y) \leftarrow & X = c \wedge Y = d & \langle 4,\langle 3\rangle\rangle
\end{array}
$$

Note that constraint atoms 3, 6, and 7 are no longer derivable since the constraint part of clause 3 in the modified program, $X = c \wedge Y = d \wedge not(X = c \wedge Y = d)$ is not solvable.

The computation of Algorithm 2 proceeds as follows. First constraint atom 3 in M is replaced by

$$P(X,Y) \leftarrow X = c \wedge Y = d \wedge not(X = c \wedge Y = d)$$

and the pair $(P(X,Y) \leftarrow X = c \wedge Y = d, \langle 3\rangle)$ is placed in P_OUT.

Next constraint atom 6 in M, due to the match within its support with the support $\langle 3\rangle$ from the above pair, is replaced by

$$A(X,Y) \leftarrow X = c \wedge Y = d \wedge not(X = c \wedge Y = d)$$

while simultaneously, the pair $(A(X,Y) \leftarrow X = c \wedge Y = d, \langle 4,\langle 3\rangle\rangle)$ is added to P_OUT.

Finally, standardizing variables apart, constraint atom 7 in M is replaced by

$$A(X,Y) \leftarrow X = X' \wedge Z = Y' \wedge X' = a \wedge Y' = c \wedge Z = X'' \wedge Y = Y'' \wedge X'' = c \wedge Y'' = d \wedge$$
$$Z = X''' \wedge Y = Y''' \wedge not(X''' = c \wedge Y''' = d).$$

Though a new pair is added to the set P_OUT, no more replacement is made to M and hence the final view is:

1. $P(X,Y) \leftarrow X = a \wedge Y = b$ $\quad\langle 1\rangle$
2. $P(X,Y) \leftarrow X = a \wedge Y = c$ $\quad\langle 2\rangle$
3. $P(X,Y) \leftarrow X = c \wedge Y = d \wedge not(X = c \wedge Y = d)$ $\quad\langle 3\rangle$
4. $A(X,Y) \leftarrow X = a \wedge Y = b$ $\quad\langle 4,\langle 1\rangle\rangle$
5. $A(X,Y) \leftarrow X = a \wedge Y = c$ $\quad\langle 4,\langle 2\rangle\rangle$
6. $A(X,Y) \leftarrow X = c \wedge Y = d \wedge not(X = c \wedge Y = d)$ $\quad\langle 4,\langle 3\rangle\rangle$
7. $A(X,Y) \leftarrow X = X' \wedge Z = Y' \wedge X' = a \wedge Y' = c \wedge Z = X'' \wedge Y = Y'' \wedge X'' = c \wedge Y'' = d \wedge$
   $Z = X''' \wedge Y = Y''' \wedge not(X''' = c \wedge Y''' = d)$ $\quad\langle 4,\langle 2\rangle,\langle 4,\langle 3\rangle\rangle\rangle$

The constraints of each of constraint atoms 3, 6, and 7 are not solvable. Hence these atoms may be removed. This produces the same materialized view as M'.

Theorem 2 *The Straight Deletion Algorithm is correct, i.e. the output M of the algorithm satisfies* $[M] = [T_{P'} \uparrow \omega(\emptyset)]$.

3.2 Insertion of Constrained Atoms

To insert the constrained atom $A(\vec{X}) \leftarrow \Theta$ into the mediated materialized view, we first construct the input *Add*, which is the set $\{A(\vec{X}) \leftarrow \mathbf{not}(\Lambda) \wedge \Theta$ such that $A(X) \leftarrow \Lambda$ is in M and $\mathbf{not}(\Lambda) \wedge \Theta$ is solvable $\}$. The set *Add* consists of all constrained atoms whose solutions correspond to the instances to be inserted into the materialized view.

Declarative Semantics: We now specify the meaning of an insertion of $A(\vec{X}) \leftarrow \Theta$ into a mediated materialized view, M, w.r.t. constrained database P – this meaning is the meaning of a constrained database $P^\flat$ constructed as follows.

$\boxed{\text{Rewrite the Constrained Database } P}$ into a *new* constrained database $P^\flat$ as follows:

$$\begin{aligned} P^\flat = P \quad & \cup \quad Add \\ & \cup \quad \{A(\vec{X}) \leftarrow \mathbf{not}(\Lambda) \wedge \Theta \mid\mid B_1(\vec{t_1}), \ldots, B_n(\vec{t_n}) \mid \\ & \qquad A(\vec{X}) \leftarrow \Gamma \mid\mid B_1(\vec{t_1}), \ldots, B_n(\vec{t_n}) \in P, n > 0, \\ & \qquad A(\vec{X}) \leftarrow \Lambda \in M\} \end{aligned}$$

Note that in the third component of the above union, for every constrained atom $A(X) \leftarrow \Lambda$ in M, and for every clause C in P with A in the head, we are replacing C's constraint part (which may have been, say, Γ) by the constraint $\mathbf{not}(\Lambda) \wedge \Theta$.

The least model of the above constrained database $P^\flat$ specifies the desired semantics after the insertion is accomplished. The reader may specifically note that even though negation occurs in the body of clauses in $P^\flat$, this negation occurs in the *constraint* part of the clause, and hence, the resulting constrained database still has a least fixpoint [25]. We now present an algorithm that incrementally inserts a constrained atom into a materialized view.

Algorithm 3 (Constrained Atom Insertion)

1. Unfold the constraint base fact to be inserted with respect to the *original* constrained database P.

$$\begin{aligned} P\_ADD_0 &= Add \\ P\_ADD_{k+1} &= P\_ADD_k \cup \{B(\vec{X}) \leftarrow \Xi \mid \\ & \quad \text{There is a clause } B(\vec{X}) \leftarrow \Xi_0 \mid\mid B_1(\vec{t_1}), \ldots, B_n(\vec{t}_n) \text{ in } P \\ & \quad \text{where for at least one } j \in \{1, \ldots, n\}, B_j(\vec{X}_j) \leftarrow \Xi_j \in P\_ADD_k, \\ & \quad \text{and for each } i \in \{1, \ldots, n\} \text{ where } B_i(\vec{X}_i) \leftarrow \Xi_i \notin P\_ADD_k \\ & \quad B_i(\vec{X}_i) \leftarrow \Xi_i \text{ is a constraint atom} \\ & \quad \text{in the materialized view } M = T_P \uparrow \omega(\emptyset), \text{ and} \\ & \quad \Xi = \Xi_0 \wedge \ldots \wedge \Xi_n \wedge (\vec{X}_1 = \vec{t_1}) \wedge \ldots \wedge (\vec{X}_n = \vec{t}_n) \\ & \quad \text{is satisfiable}\} \\ P\_ADD &= P\_ADD_\omega \end{aligned}$$

2. Set $M' = M \cup P\_ADD$, which is then the new view.

Observe that an important difference between the deletion and the insertion algorithms is that in the condition defining P_Add_{k+1}, the number of body literals B_i that are contained in P_ADD_k is one *or more*. Recall that in the construction of P_OUT_{k+1}, we require that the number of body literals contained in P_OUT_k is exactly one.

The next theorem establishes the correctness of this algorithm, i.e. the incrementally computed view, M', is the same as the least fixpoint of $T_{P^\flat}$ where $P^\flat$ is the rewritten constraint database.

Theorem 3 The insertion algorithm is correct. i.e. $[T_{P^\flat} \uparrow \omega(\emptyset)] = [M']$.

4 Maintaining Views when External Changes Occur

Suppose we consider a mediator that integrates information in domains $\Sigma_1, \ldots, \Sigma_n$. For instance, these domains may be relational database systems like `PARADOX` or `DBASE`, or non-traditional systems like the `facedb` and `spatialdb` domains specified in the law enforcement example. When an update occurs *within* one or more of the domains being integrated (e.g. a `PARADOX` table gets updated), this could be viewed as a modification of the *behavior* of the functions that access these domains. For instance, the `select` function in the `PARADOX` domain may return a new set of tuples (after the update of the `PARADOX` tables). Another possibility is that the code implementing functions may also have been updated (e.g. to remove bugs in older versions of the software package). In this section, we analyze how updates to the integrated domains may affect the materialized mediated view and how they can be handled efficiently. For this, it is important to always remember that we do not materialize the functions occurring within the `in` predicate but instead materialize the mediated view by unfolding its defining rules.

As the behavior of functions is changing over time, we will use $d : f_t$ to denote the behavior of the function f of domain d at time t. In order to capture the behavioral difference of f between two successive time points, we define

$$\Delta f^{+}_{t,t+1}(< args >) = f_{t+1}(< args >) - f_t(< args >) \tag{6}$$

$$\Delta f^{-}_{t,t+1}(< args >) = f_t(< args >) - f_{t+1}(< args >) \tag{7}$$

Thus, $\Delta f^{+}_{t,t+1}(< args >)$ is the set of values returned by executing function f at time $t+1$ that were not returned when f was executed at time t. Likewise, $\Delta f^{-}_{t,t+1}(< args >)$ is the set of objects returned by executing function f at time t that are not returned when f is executed at time $t+1$. Note that the efficient computation of the difference between two successive database states has been extensively studied [8, 28, 30, 23, 38]. However, as we will see, we do not need the difference explicitly for our view maintenance mechanism. We only use it to investigate the effects of an update to an external function onto a materialized mediated view if T_P is used.

For a constraint atom to be introduced into the materialized mediated view defined by T_P, we require that the constrain be be satisfiable; hence, we should not be surprised that the materialized mediated view changes if the functions invoked within `in` change. In order to investigate the implied changes in a little more detail, let $REM = \{\mathtt{in}(\mathtt{a}, \mathtt{d} : \mathtt{f}(\mathtt{b})) \mid \mathtt{a} \in \Delta\mathtt{f}^{-}_{\mathtt{t},\mathtt{t}+1}\}$ and $ADD = \{\mathtt{in}(\mathtt{a}, \mathtt{d} : \mathtt{f}(\mathtt{b})) \mid \mathtt{a} \in \Delta\mathtt{f}^{+}_{\mathtt{t},\mathtt{t}+1}\}$. Then, intuitively, we may regard the problem of function updates as being equivalent to the insertion and the deletion of the ground instance that correspond to the DCA-atoms in the sets ADD and REM, respectively. However, as we are working with non-ground constrained atoms, the situation is less straightforward.

The set ADD does not introduce any technical complications. In contrast, the set REM needs to be treated with care. The following example provides an illustration.

Example 7 Suppose we have a constrained database that contains the single clause

$$B(X) \leftarrow in(X, d : g(b)).$$

The function g is a call in the domain d. Assuming the initial set of values returned by g for the argument b is the singleton $\{a\}$, then according to the definition of T_P, we would have the constraint atom $B(X) \leftarrow in(X, d : g(b))$ in the original materialized view. Now suppose at time $t+1$, the result a is removed from the output of g. So $g(b) = \emptyset$. According to T_P, the materialized view at $t+1$ would be empty since the constraint $in(X, d : g(b))$ is unsolvable.

This example illustrates that the set REM may cause subsequent modifications in the materialized view. However, the requirement that changes in functions of constraint domains be reflected in the materialized view appears to only incur computational overhead with little theoretical benefits. *A better approach is to regard the materialized view as a syntactic construct where each constraint atom $A(X) \leftarrow \Xi$ defines an access into the set of solutions represented by Ξ.* In particular, if f occurs in the constraint Ξ, then at time t f will be interpreted as if it denotes the function f_t; at time $(t+1)$ it will denote the function f_{t+1}. Then, we may eliminate, from the definition of T_P, the condition that constraints be satisfiable, and instead, may defer the satisfiability test to query-evaluation time. As we demonstrate shortly, the elimination of the requirement that the constraint Ξ is satisfiable will simplify immensely the updating process. *Indeed, maintaining a materialized view requires no action whatsoever when this point of view is adopted, even if external changes occur.* We first adapt the operator T_P to the following simpler version, called W_P.

$$\begin{array}{rcl} W_P(I) & = & \{A(\vec{X}) \leftarrow \Xi \mid \\ & & \text{There is a clause } A(t_0) \leftarrow \Xi_0 || A_1(t_1), \ldots, A_n(t_n) \text{ in P} \\ & & \forall 1 \le i \le n \; : \; \exists A_i(X_i) \leftarrow \Xi_i \in I, \\ & & \text{which share no variables and the constraint } \Xi \text{ is} \\ & & \Xi_0 \wedge \Xi_1 \wedge \ldots \wedge \Xi_n \wedge \{\vec{X}_1 = \vec{t}_1\} \wedge \ldots \wedge \{\vec{X}_n = \vec{t}_n\} \wedge \{\vec{X} = \vec{t}_0\}\} \end{array}$$

Observe that the only difference between W_P and T_P is that the constraint Ξ is not required to be solvable. The materialized view of a constrained database is defined to be $W_P \uparrow \omega(\emptyset)$. Given now that the materialized view is only a syntactic construction where constraints that appear in constraint atoms are not necessarily solvable, it is clear to see that no changes to the solution sets of functions in any constraint domain will affect the syntactic form of the materialized view, as proved in the next theorem.

Theorem 4 Suppose M_t is the materialized view of the constrained database P at time point t. Then M_{t+1}, the materialized view of the constrained database P at time point $t+1$, is syntactically identical to M_t.

The *reason for this is that when we construct our materialized mediated views, we are storing atoms in the form $A \leftarrow \Xi$ where Ξ may contain some external function calls (let's say f is one* *such external call). At time t, the syntactic symbol f occurring in Ξ denotes the function f_t, i.e. it denotes the behavior of function f at time t. At time $t+1$, the syntactically identical constraint Ξ is evaluated with the syntactic entity f interpreted as the function f_{t+1}.* The reason this approach works with W_P and not with T_P is that T_P determines solvability of constraints at time t, which means that the meaning, f_t of functions at time t may be used to "eliminate" constrained atoms from the materialized view. In contrast, when no such eliminations are performed, as in the case of W_P, we can use the same syntactic form because evaluation of solvability of constraints is done using the "current meaning" of f, i.e. the meaning of f at time $t+1$.

Hence no action is required in view maintenance as the result of changes to domain functions. More important than the fact that the syntactic form of the materialized view remains static is that semantically, the instances represented by this single view accurately reflects the instances that should be true for the given constrained database at any time point. More specifically, the instances of the view that is constructed using W_P will coincide with the instances of the view constructed using T_P.

Corollary 1 Let $M = W_P \uparrow \omega(\emptyset)$. Suppose M_t represents the materialized view of the constrained database constructed using T_P and where the function calls to domains are evaluated at time point t, for any t. Then $[M] = [M_t]$.

Example 8 Let P contain the single rule $A(X) \leftarrow in(X, \Sigma_1 : f(X)) \mid\mid B(X,Y)$ and the two facts $\{B(a,b), B(b,b)\}$. Suppose the function f evaluated at time t behaves as $f_t(b) = \{b\}$ and $f_t(X) = \emptyset$ for $X \neq b$. The materialized view M constructed under W_P is

$$\begin{aligned}\{\ & B(a,b), B(b,b),\\ & A(X) \leftarrow in(X, \Sigma_1 : f(X)) \wedge X = a \wedge Y = b\\ & A(X) \leftarrow in(X, \Sigma_1 : f(X)) \wedge X = b \wedge Y = b\}\end{aligned}$$

and its instances $[M]$ is the set $\{B(a,b), B(b,b), A(b)\}$. Using T_P, M_t is identical to M with the exception that it does not contain the third constraint atom as listed above for M. Clearly, $[M] = [M_t]$.

Now suppose the behavior of f at time $t+1$ is $f_{t+1}(a) = a$ and $f_{t+1}(X) = \emptyset$ for $X \neq a$. M remains unchanged while the new materialized view according to T_P will be $M - \{A(X) \leftarrow in(X, \Sigma_1 : f(X)) \wedge X = b \wedge Y = b\}$. Again, we have $[M] = [M_{t+1}]$ which is now the set $\{B(a,b), B(b,b), A(a)\}$.

5 Discussion

Materialization of mediated views is performed by unfolding the rules defining the view. An update of kind one, that is an update to the view, invalidates the materialized mediated view but — in our case — is *not* propagated to the integrated domains as incorporated by the `in` predicate. This makes our approach different from work on view updates on relational, deductive and object-oriented databases as partially cited and discussed in the introduction of this paper. Note, that none of this work is based on a language as powerful as constrained logic. However, considering the orthogonality of the approaches, it might be worthwhile to investigate an integration of this work with our approach. To some extent, this has already been done in this paper – for instance, the DRed algorithm presented in [22] has been extended to handle deletions in constrained and mediated databases. The relationship between the DRed algorithm and algorithms in [11, 12, 44, 30] has been discussed in detail in [22] – however, none of these algorithms deal with constraints, and they all assume that a materialized view contains only *ground, fully instantiated* tuples – assumptions that are removed in this paper.

As we have seen, an update of the second kind — a change to one of the integrated domains — affects the materialized mediated view if the T_P fixpoint operator is used. By replacing if by W_P, we could eliminate the implied recomputation. Again, this differs from the traditional approach to view maintenance, since only the unfolding process of the rules which is *independent* on the actual evaluation of the `in` predicate might be affected. But even this effect, forcing the recomputation for T_P is eliminated by using W_P while preserving the semantics of T_P. However, the work on view maintenance which was partially cited in the introduction of this paper becomes relevant as soon as we want to guarantee an efficient evaluation of the `in` predicate by materializing the external function calls. Especially for the integration of software packages, the methods presented in [28] become relevant.

6 Conclusion and future work

It is generally well accepted that *constrained databases* are very important, increasing the expressive power of Datalog considerably. In this paper, we have developed techniques to incrementally update views in constrained databases. The HERMES system at the University of Maryland is based on the intuition that constraints can be used to integrate multiple databases, multiple data structures, and multiple reasoning paradigms. In its current form, HERMES integrates INGRES, PARADOX, path planning packages developed by the US Army, Face Recognition packages used in Federal Law Enforcement, spatial data structures, a text database, and a pictorial database. Descriptions of the theory of HERMES may be found in [40, 4, 5, 39, 40, 41] – in particular, [26] shows that HERMES generalizes constrained databases as proposed by Kanellakis et. al. [27].

In this paper, we have dealt with the problem of efficiently maintaining materialized mediated views such as those that may occur when any constrained database system is updated. To our knowledge, this is the first paper that addresses the view maintenance problem for constrained databases and/or for heterogeneous, mediated systems. The main contributions we have made are the following:

- We have shown how the DRed deletion algorithm of Gupta et. al. [22] may be extended to handle constraints.
- We have developed a unique *straight delete* algorithm for deletion that uses *supports* to accomplish deletions of constrained atoms; this algorithm is brand new, and, even when constraints are absent, it improves upon the counting method (that can lead to infinite counts) [21] and also improves upon the re-derivation algorithm (as it requires no rederivations. In addition, as shown in the paper, it also applies to databases with constraints in it, including mediated systems.
- We have developed algorithms for inserting *constrained* atoms into an existing materialized view.

- We have shown that when we eliminate the constraint-satisfiability check from the Gabbrielli-Levi operator, then the problem of maintaining views in mediated systems (when changes occur in different programs/databases participating in the mediated framework) can be handled very easily indeed – no change to the mediated view, whatsoever, is needed, when the notion of mediated view defined by W_P is adopted ! This makes our approach eminently suitable for mediated systems.

References

[1] S. Abiteboul and A. Bonner. Objects and view. In *Proc. of the ACM SIGMOD Conf. on Management of Data*, pages 238–247, 1991.

[2] S. Abiteboul, S. Cluet, and T. Milo. Querying and updating the file. In *Proc. Int. Conf. on Very Large Data Bases (VLDB)*, pages 73–84, 1993.

[3] S. Adalı and R. Emery. (1994) *A Uniform Framework for Integrating Knowledge in Heterogeneous Knowledge Systems*, to appear in *Proceedings of ICDE.*

[4] S. Adalı and V.S. Subrahmanian. (1993) *Amalgamating Knowledge Bases, II: Algorithms, Data Structures and Query Processing*, submitted for journal publication.

[5] S. Adalı and V.S. Subrahmanian. (1994) *Amalgamating Knowledge Bases, III: Distributed Mediators*, to appear in *Journal of Cooperative Information Systems*, Dec. 1994.

[6] F. Bancilhon and N. Spyratos. Update semantics and relational views. *ACM Trans. on Database Systems*, 6(4):557–575, 1981.

[7] M. Baudinet, M. Niezette, P. Wolper. (1991) *On the Representation of Infinite Temporal Data and Queries*, in Proceedings of the Tenth ACM Symposium on Principles of Database Systems, Denver, Colorado, May 1991, pp.280-290

[8] J. Blakeley, N. Coburn, and P.-A. Larson. Updating derived relations: Detecting irrelevant and autonomously computable updates. *ACM Trans. on Database Systems*, 14(3):369–400, 1989.

[9] J. Chomicki. (1990) *Polynomial Time Query Processing in Temporal Deductive Databases*, in Proceedings of the Ninth ACM Symposium on Principles of Database Systems, Nashville, Tennessee, April 1990, pp. 379-391

[10] Communications of the ACM 37 (7). Intelligent agents (Special Issue), July 1994.

[11] Stefano Ceri and Jennifer Widom. Deriving Production Rules for Incremental View Maintenance. In *17th VLDB*, 1991.

[12] Stefano Ceri and Jennifer Widom. Deriving Incremental Production Rules for Deductive Data. IBM RJ 9071, IBM Almaden, 1992.

[13] C.C. Cosmadakis and C.H. Papadmitriou. Updates of relational views. *Journal of the ACM*, 31(4):742–760, 1984.

[14] U. Dayal. Queries and views in a object-oriented databases. In *Int. Workshop on Database Programming Languages*, 1989.

[15] U. Dayal and P. Bernstein. On the correct translation of update operations on relational views. *ACM Trans. on Database Systems*, 7(3), 1982.

[16] U. Dayal and H. Hwang. View definition and generalization for database integration in a multi-database system. *IEEE Trans. Software Eng.*, SE-10(6):628–644, 1984.

[17] J. Doyle. A Truth Maintenance System. Artificial Intelligence, vol. 12, 1979.

[18] M. Falaschi, G. Levi, M. Martelli, C. Palamidessi (1991) *A New Declarative Semantics for Logic Programs*, ICLP, 1991

[19] M. Gabbrielli, G. Levi. (1991) *Modelling answer constraints in Constraint Logic Programs*, ICLP, 1991, pp.238-251,

[20] N. Gehani, H. Jagadish, and W. Roome. OdeFS: A file system interface to an object-oriented database. In *Proc. Int. Conf. on Very Large Data Bases (VLDB)*, pages 249–260, 1994.

[21] Ashish Gupta, Dinesh Katiyar, and Inderpal Singh Mumick. Counting Solutions to the View Maintenance Problem. In *Workshop on Deductive Databases, JICSLP*, 1992.

[22] A. Gupta, I.S. Mumick and V.S. Subrahmanian. Maintaining Views Incrementally, *Proc. 1993 ACM SIGMOD Conf. on Management of Data*, Washington, DC.

[23] E. Hanson. A performance analysis of view materialization strategies. In *Proc. of the ACM SIGMOD Conf. on Management of Data*, pages 440–453, 1987.

[24] John V. Harrison and Suzanne Dietrich. Maintenance of Materialized Views in a Deductive Database: An Update Propagation Approach. In *Workshop on Deductive Databases, JICSLP*, 1992.

[25] J. Jaffar, J.-L. Lassez. *Constraint Logic Programming* In. Proceedings of Fourteenth Annual ACM Symposium on Principles of Programming Languages, pp. 111-119. ACM, New York, USA, 1987

[26] J. Lu, A. Nerode and V.S. Subrahmanian. *Hybrid Knowledge Bases*, to appear in *IEEE Transactions on Data and Knowledge Engineering.*

[27] P. Kanellakis, G. Kuper and P. Revesz. (1990) *Constraint Query Languages*, Proc. 9th ACM Symp. on Principles of Database Systems, pps 299-313.

[28] A. Kemper, C. Kilger, G. Moerkotte. *Function Materialization in Object Bases: Design, Realization, and Evaluation* IEEE Transactions on Knowledge and Data Engineering, Vol.6, No.4, August 1994

[29] R. Krishnamurthy, W. Litwin, and W. Kent. Language features for interoperability of databases with schematic discrepancies. In *Proc. of the ACM SIGMOD Conf. on Management of Data*, pages 40–49, 1991.

[30] V. Küchenhoff. On the efficient computation of the difference between consecutive database states. In *Proc. Int. Conf. on Deductive and Object-Oriented Databases (DOOD)*, pages 478–502, 1991.

[31] C. Medeiros and J.-C. Mamou. Interactive manipulation of object-oriented views. In *Proc. IEEE Conference on Data Engineering*, 1991.

[32] G. Moerkotte and P.C. Lockemann. Reactive consistency control in deductive databases. *ACM Trans. on Database Systems*, 16(4):670–702, 1991.

[33] A. Motro. Superviews: Virtual integration of multiple databases. *IEEE Trans. Software Eng.*, 13(7):785–798, 1987.

[34] I. S. Mumick. *Query Optimization in Deductive and Relational Databases.* Ph.D. Thesis, Stanford University, CA 94305, 1991.

[35] M. Scholl, C. Laasch, and M. Tresch. Updatable views in object-oriented databases. In *Proc. Int. Conf. on Deductive and Object-Oriented Databases (DOOD)*, 1991.

[36] A. Sheth and J. Larson. Federated database systems for managing distributed, heterogeneous and autonomous databases. *ACM Computing Surveys*, 22(3):183–236, 1990.

[37] Oded Shmueli and Alon Itai. *Maintenance of Views.* In *Sigmod Record*, 14(2):240-255, 1984.

[38] M. Stonebraker, A. Jhingran, J. Goh, and S. Potamianos. On rules, procedures, caching and views in data base systems. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 281–290, 1990.

[39] V.S. Subrahmanian. *Amalgamating Knowledge Bases, ACM Trans. on Database Systems*, 19, 2, pps 291–331, 1994.

[40] V.S. Subrahmanian. *Hybrid Knowledge Bases for Integrating Symbolic, Numeric and Image Data*, Proc. 1994 Intl. Workshop on Applied Imagery and Pattern Recognition, Washington DC, Oct. 1994, SPIE Press.

[41] V.S. Subrahmanian, S. Adali, A. Brink, R. Emery, A. Rajput, T.J. Rogers, R. Ross, and C. ward. (1994) *HERMES: A Heterogeneous Reasoning and Mediator System*, draft manuscript.

[42] E. Teniente. Updating knowledge bases while maintaining their consistency. *VLDB Journal*, 1994. to appear.

[43] A. Tomasic. View update translation via deduction and annotation. In *ICDT*, pages 338–352, 1988.

[44] Toni Urpi and Antoni Olive. A Method for Change Computation in Deductive Databases. In *18th VLDB*, pages 225–237, 1992.

[45] G. Wiederhold. (1993) *Intelligent Integration of Information*, Proc. 1993 ACM SIGMOD Conf. on Management of Data, pps 434–437.

[46] G Wiederhold, S. Jajodia, and W. Litwin. Integrating temporal data in a heterogeneous environment. In *Temporal Databases.* Benjamin/Cummings, Jan 1993.