Canonical View Update Support through Boolean Algebras of Components

Stephen J. Hegner
Department of Computer Science and Electrical Engineering
Votey Building
University of Vermont
Burlington, VT 05405

(802) 656-3330

January 1984

This work was supported by the National Science Foundation under grant IST82-12178.
ABSTRACT

The problem of reflecting updates from a view schema to the base schema in relational database systems is investigated. The basic strategy which is employed is that of translation relative to a constant complement, as developed by Bancilhon and Spyropoulos. We deal in particular with one of the apparent drawbacks of the constant complement approach, namely, the nonuniqueness of such complements. Our approach is to use only a specific set of well-behaved views, termed the components of the schema, as complements. The components form a Boolean algebra, and update reflection to the base schema is independent of choice of complement, as long as the complement is a component. We argue that such reflections of updates are the "natural" ones which should be allowed.
0. INTRODUCTION

0.1 Basic Problem Statement

Avoiding technical details for the moment, we think of a relational database schema \( \mathbf{V} \) as a pair \((\text{Rel}(\mathbf{V}), \text{Con}(\mathbf{V}))\), where \(\text{Rel}(\mathbf{V})\) is a finite set of relation symbols, and \(\text{Con}(\mathbf{V})\) is a set of integrity constraints. A legal database for \(\mathbf{V}\) is just an indexed set of relations, one for each \(R \in \text{Rel}(\mathbf{V})\), which collectively satisfy the integrity constraints of \(\text{Con}(\mathbf{V})\). Denote the collection of all legal databases of \(\mathbf{V}\) by \(\text{LDB}(\mathbf{V})\).

0.1.1 DEFINITION  An update specification for \(\mathbf{V}\) is a pair \((s_1, s_2)\), where \(s_1\) and \(s_2\) are each elements of \(\text{LDB}(\mathbf{V})\). Think of \(s_1\) as the current state of the schema, and \(s_2\) as the desired next state. Insertions, deletions, and replacements are commonly considered special cases of this general notion of update.

In a relational database system, we assume that there is a main relational schema \(\mathbf{D}\), designated the base schema. This main schema encompasses the entire "enterprise" which is modelled, in the sense of the conceptual schema of the ANSI/SPARC proposal [ANSI75]. It is further assumed that any update specification \((s_1, s_2)\) on this schema is supported (at least in principle), although certain special privileges may be required to effect such an update. In short, for the purposes of this paper, we consider the update problem on the base schema to be solved.

For reasons of security and ease of use, a user is not generally given direct access to the base schema. Rather, his window on the system is via a
view. A view is given by a pair \( \Gamma = (V, \gamma) \), where \( V \) is a relational database schema, and \( \gamma : D \rightarrow V \) is a database mapping. Such a mapping defines a function \( \gamma' : \text{LDB}(D) \rightarrow \text{LDB}(V) \), assigning to each legal state \( s \in \text{LDB}(D) \) a state \( \gamma'(s) \in \text{LDB}(V) \). (The reason for the "'" notation will become apparent later.)

Regardless of how the system consisting of the base schema and collection of user views is actually implemented, it is necessary to maintain global consistency, in the sense that the state of each view \( \Gamma = (V, \gamma) \) should always be the image \( \gamma'(s_1) \) of the state \( s_1 \) of the base schema. Whenever an update \( (s_1, s_2) \) is specified directly on the base schema \( D \), this is no problem, as the new state of view \( \Gamma \) is just \( \gamma'(s_2) \). However, when an update \( (t_1, t_2) \) is specified by a user directly on \( \Gamma \), there is a problem of determining an appropriate next state for the base schema. The following definitions formalize the situation.

0.1.2 DEFINITION Let \( D \) be a database schema, and let \( \Gamma = (V, \gamma) \) be a view of \( D \).

(a) An update specification for \( \Gamma \) is a pair \( (s_1, (t_1, t_2)) \), where \( (t_1, t_2) \) is an update specification to the schema \( V \) and \( s_1 \in \text{LDB}(D) \) is such that \( \gamma'(s_1) = t_1 \). (Think of \( s_1 \) as the current state of the base schema, and \( (t_1, t_2) \) as the update request of the user.)

(b) A solution to the update specification \( (s_1, (t_1, t_2)) \) is an \( s_2 \in \text{LDB}(D) \) such that \( \gamma'(s_2) = t_2 \).

(c) An update strategy for \( \Gamma \) is a (possibly partial) function \( \rho : \text{LDB}(D) \times \text{LDB}(V) \rightarrow \text{LDB}(D) \). Think of
\( \rho : (\text{current state of } D, \text{ desired new state of } V) \rightarrow \text{ new state of } D, \text{ if the update is allowed}; \)

\[ \text{undefined if the update is not allowed.} \]

Thus, for each update specification \((s_1, (t_1, t_2))\), \(\rho\) gives the solution \(\rho(s_1, t_2)\), whenever this value is defined.

0.2 Our Approach in Relation to Existing Results

The problem of determining an appropriate update strategy has been addressed by a number of researchers. Most of the works concern situations in which the base schema is constrained only by simple constraints, such as functional and/or inclusion dependencies ([DaBe78], [DaBe82], [CaAr79], [FuSS79], [Kell82], [CoPa83]). While these works have provided valuable insight into the way in which updates may be reflected, we are interested in establishing general principles of view support, independent of the detailed properties of specific database constraints and mappings.

A much more general approach, which forms the roots of our work, is the constant complement approach of Bancilhon and Spyropoulos ([Banc79], [BaSp81a], [BaSp81b]). A join complement to view \(\Gamma_1\) is another view \(\Gamma_2\) such that the combined states of the two views uniquely determine the state of the base schema. In this situation, it is easy to see that for any update specification \((s_1, (t_1, t_2))\) for \(\Gamma_1\), there can be at most one solution which leaves the state of \(\Gamma_2\) fixed, so the selection of such a \(\Gamma_2\) determines uniquely a maximal update strategy on \(\Gamma_1\). The problem is, of course, how to pick \(\Gamma_2\). In general, there are many join complements to \(\Gamma_1\), and the resulting strategy depends upon the choice. Our work picks up where that of Bancilhon and
Spyratos leaves off, in the sense that we develop a "natural" set of views from which the join complements should be chosen. This set of views forms a Boolean algebra under certain natural operations, and is termed the algebra of components. We show that by using only members of the algebra of components as join complements, update strategies are unique and independent of the particular complement chosen.

0.3 Outline of the Paper

In section 1, we present a general analysis of the update problem, together with general conditions which any update strategy ought to meet. Also presented is a more detailed development of the constant complement strategy. For this section, we presume a general knowledge of the commonly used language of database theory, such as can be found in [Ullm82] and [Maie83], but no other specialized knowledge.

In section 2, we outline the necessary features of our general algebraic framework for decomposition, which is developed in detail in [Hegn84]. This framework is the cornerstone of our theory, as the Boolean algebra of components is precisely the set of components which arise as the basic units in our decomposition theory.

In section 3, we present the decomposition results, based upon constant complements consisting of components.
1. **SURVEY OF PROBLEM CHARACTERISTICS**

1.1 The Surjectivity Problem

The issues involved in the surjectivity problem are best illustrated via a simple example.

1.1.1 EXAMPLE Let $D$ be the base schema with two binary relation symbols $R_{SP}$ and $R_{PJ}$, and no constraints whatever. Let $V$ be the view with the single ternary relation symbol $R_{SPJ}$, and let $\gamma$ be the view mapping which defines the join of $R_{SP}$ and $R_{PJ}$ on the attribute $P$. A typical instance of this schema and view is shown below.

<table>
<thead>
<tr>
<th>$S$ : $P$</th>
<th>$P$ : $J$</th>
<th>$S$ : $P$ : $J$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1$ : $P_1$</td>
<td>$P_1$ : $J_1$</td>
<td>$s_1$ : $P_1$ : $J_1$</td>
</tr>
<tr>
<td>$s_1$ : $P_2$</td>
<td>$P_1$ : $J_2$</td>
<td>$s_1$ : $P_1$ : $J_1$</td>
</tr>
<tr>
<td>$s_2$ : $P_3$</td>
<td>$P_3$ : $J_1$</td>
<td>$s_2$ : $P_3$ : $J_1$</td>
</tr>
<tr>
<td>$P_4$ : $J_3$</td>
<td>$R_{SP}$</td>
<td>$R_{PJ}$</td>
</tr>
</tbody>
</table>

Now suppose that the update which inserts the tuple $(s_3, P_3, J_3)$ into the view instance is requested, so that the new state should be instance (a) below. It is easy to see that there is no state of the schema $D$ which has this instance as this image, because $R_{SPJ}$ is a join, and hence any image under $\gamma'$ must satisfy the join dependency $*[SP, PJ]$. To insert this new tuple, we would have to insert $(s_3, P_3)$ into $R_{SP}$ and $(P_3, J_3)$ into $R_{PJ}$, which would force the instance of $R_{SPJ}$ to be (b) below.
The insertion of \((s_3, p_3, j_1)\) and \((s_2, p_3, j_3)\) into the view is termed a side effect in [DaBe78]. In the terminology of [DaBe82], the update has been performed, but not performed exactly.

The problem here is that we have not endowed the user view with the constraints inherited from the base view. An implied constraint of view \(\Gamma = (V, \gamma)\) is a constraint on \(V\) which is true for every instance of the form \(\gamma'(s)\), with \(s \in LDB(D)\). In this case, \(\mathcal{W}(SP, PJ)\) is an implied constraint of the view, and should be a consequence of \(\text{Con}(V)\).

In this work, we shall always assume that the instance mapping \(\gamma' : LDB(D) \rightarrow LDB(V)\) of a view \(\Gamma = (V, \gamma)\) is surjective. In other words, the constraints of \(V\) are sufficient to forbid any view states which are not images of base schema states. Therefore, it will never be the case that no reflection exists for a given update specification of the view.

Let us remark that it is not always possible to guarantee the surjectivity of a view mapping simply by including first-order implied constraints, even when the base schema is constrained only by functional dependencies, and the view consists entirely of projections. For an example of this situation, consult [Hegn84]. Also, for an elegant and thorough study of the implied constraint problem, see [JaAK82].
1.2 The Admissibility Problem

The admissibility problem is one of selecting an appropriate update strategy. In this section, we present several requirements which any such strategy ought to possess.

REQUIREMENT 1: An admissible update strategy must not introduce any extraneous components into the new state of the base schema.

The concept of extraneous update is borrowed from [DaBe78]. Intuitively, an extraneous component of an update reflection is one which is unnecessary to achieve the intended result. An example will make the idea clear.

1.2.1 EXAMPLE Consider again the base and view schemata and example instance of Example 1.1.1. Suppose that we wish to implement the deletion of the tuple \((s_1, p_1, j_1)\) from the view relation \(R_{SPJ}\). This can be accomplished by removing \((p_1, j_1)\) from the instance of \(R_{PJ}\). However, the tuple \((p_4, j_3)\) could also be removed from the instance of \(R_{PJ}\), without altering the effect. The deletion of \((p_4, j_3)\) is an extraneous update, since it is totally unnecessary to achieve the desired effect.

The idea that the reflected changes should be minimal is espoused in [Kell82], and is also central in the general work on the semantics of update in [FaUV83]. The following example illustrates the general idea.

1.2.2 EXAMPLE Consider again the general setting of Example 1.1.1. Suppose
this time that we wish to delete the tuple \((s_2, p_3, j_1)\) from the view relation instance \(R_{SPJ}\). This can be accomplished by deleting either \((s_2, p_3)\) from the instance of \(R_{SP}\) or by deleting \((p_3, j_1)\) from the instance of \(R_{PJ}\), or both. There is no minimal change which will realize the desired update, and so we must either make an arbitrary choice, or else delete both tuples. In any case there is no minimal operation on the base state which will accomplish the intended result. However, each of these updates is nonextraneous, as defined below.

1.2.3 NOTATIONAL CONVENTION Given a relational database schema \(D\), we define the set operations \(\subseteq, \cap, \cup, \setminus\) (difference), and \(\Delta\) (symmetric difference) in a relation-by-relation fashion. (Recall that \(A \Delta B = (A \cup B) \setminus (A \cap B)\).)

1.2.4 DEFINITION Let \(\Gamma = (\mathbf{V}, \mathbf{Y})\) be a view of schema \(D\). (a) Let \((s_1, (t_1, t_2))\) be an update specification for \(\Gamma\). A solution \(s_2\) to \((s_1, (t_1, t_2))\) is nonextraneous (resp. minimal) if for any other solution \(s_3\), \(s_1 \Delta s_3 \subseteq s_1 \Delta s_2\) (resp. \(s_1 \Delta s_2 \subseteq s_1 \Delta s_3\)).

(b) An update strategy \(\varnothing\) for \(\Gamma\) is nonextraneous (resp. minimal) if each of its associated defined update strategies is nonextraneous (resp. minimal).

It is tempting to require that an update strategy always have minimal solutions. Unfortunately, this is not possible, as the following example shows.

1.2.5 EXAMPLE Let us, in effect, "turn around" Example 1.1.1. Let \(D\) be the schema with the single relation symbol \(R_{SPJ}\). Mandate that \(R_{SPJ}\) satisfy the join dependency \(M(SP, PJ)\). Let \(\Gamma_1\) be the view which is defined by the
projection $\pi_{SP}$ to produce the relation $R_{SP}$. Similarly, let $\Gamma_2$ be the view which is defined by the projection $\pi_{PJ}$ to produce the relation $R_{PJ}$. Suppose that the initial states of the schemata are

<table>
<thead>
<tr>
<th>S</th>
<th>P</th>
<th>J</th>
</tr>
</thead>
<tbody>
<tr>
<td>s_1</td>
<td>p_1</td>
<td>j_1</td>
</tr>
<tr>
<td>s_1</td>
<td>p_1</td>
<td>j_2</td>
</tr>
<tr>
<td>s_2</td>
<td>p_2</td>
<td>j_2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>s_1</td>
<td>p_1</td>
</tr>
<tr>
<td>s_2</td>
<td>p_2</td>
</tr>
</tbody>
</table>

Now suppose that we wish to insert $(s_3, p_1)$ into the relation of view $\Gamma_1$. The most obvious way to do this is to insert $(s_3, p_1, j_1)$ and $(s_3, p_1, j_2)$ into the base relation. However, we could also insert just $(s_3, p_1, j_1)$ into the base relation, and delete $(s_1, p_1, j_2)$. Neither update is extraneous, and so no minimal solution is possible in this case.

We do have the following weaker result on minimal solutions.

1.2.6 PROPOSITION Let $\rho$ be an update strategy, and suppose that $\rho$ always supplies nonextraneous solutions. Then, whenever a minimal solution exists, $\rho$ will supply that minimal solution.

Proof: Follows immediately from the fact that if a minimal solution to an update problem exists, then it is the only nonextraneous solution. ■

REQUIREMENT 2: An admissible update strategy must be functorial.

1.2.7 EXAMPLE We again use the schema, view, and initial instances of Example 1.1.1. Suppose that our first update request is to change $(s_2, p_3, j_1)$ to $(s_2, p_4, j_1)$ in the view instance. To do so, we change $(s_2, p_3)$ to $(s_2, p_4)$ in the instance of $R_{SP}$, and change $(p_3, j_1)$ to $(p_4, j_1)$ in the instance of $R_{PJ}$. To avoid having $(s_2, p_4, j_3)$ appear in the view, $(p_4, j_3)$ must also be deleted from
the instance of $R_{pj}$. It is clear that this defines a minimal update which achieves the desired result. Now suppose that we wish to change $(s_2, p_4, j_1)$ back to $(s_2, p_3, j_1)$. To achieve this, we change $(s_2, p_4)$ back to $(s_2, p_3)$ in the instance of $R_{sp}$, and change $(p_4, j_1)$ back to $(p_3, j_1)$ in the instance of $R_{pj}$. Again, this is a minimal update. Note, however, that the resulting state of the base schema is not the same as the initial state; the result of applying these two changes (which amount to the identity update on the view) does not make the same change on the base view as the identity update (which would make no changes at all to the base instance). Thus, any update strategy with these changes is not functorial in the sense of the following definition.

1.2.8 DEFINITION Let $\Gamma = (V, \gamma)$ be a view of the schema D, and let $\rho$ be an update strategy for $\Gamma$. $\rho$ is functorial if the following two conditions are satisfied.

(a) If $s_1 \in LDB(D)$ and $t_2 \in LDB(V)$ are such that $\gamma'(s_1) = t_2$, then $\rho(s_1, t_2) = s_1$.

(b) If $s_1 \in LDB(D)$ and $t_2, t_3 \in LDB(V)$ are such that $\rho(s_1, t_2)$ and $\rho(\rho(s_1, t_2), t_3)$ are defined, then $\rho(s_1, t_3) = \rho(\rho(s_1, t_2), t_3)$.

Condition (a) of functoriality just says that the identity update should be reflected in the base schema as no change. Condition (b) says that if we start in base state $s_1$ and view state $t_1$, going first to view state $t_2$ and then to $t_3$ results in the same base schema state as updating directly from $t_1$ to $t_3$. The key property of functoriality is summarized in the following.

1.2.9 OBSERVATION Let $\Gamma = (V, \gamma)$ be a view of the schema D, and let $\rho$ be an update strategy for $\Gamma$. If $\rho$ is functorial, then for any initial state of the
base schema and sequence of updates on the view, the final state of the base
schema depends only upon the final state of the view schema, and not upon the
sequence of updates applied to get there. 

REQUIREMENT 3: An admissible update strategy must be symmetric.

1.2.10 EXAMPLE Again consider the base schema, view, and initial instances of
Example 1.1.1. Suppose we first wish to insert \((s_1, p_4, j_4)\) into the view. To
do so minimally, we insert \((s_1, p_4)\) into the instance of \(R_{SP}\), insert \((p_4, j_4)\)
to the instance of \(R_{PJ}\), and delete \((p_4, j_3)\) from the instance of \(R_{PJ}\). Now
suppose we wish to delete this same tuple from the view. We can delete either
\((s_1, p_4)\) or \((p_4, j_4)\) from the appropriate relation to get a nonextraneous dele-
tion. If we seek an update strategy which only allows nonextraneous updates,
and which allows the insertion, then the strategy is not symmetric, since the
effect of the insertion cannot be undone. This is formalized in the next
definition.

1.2.11 DEFINITION Let \(\Gamma = (V, \gamma)\) be a view of schema \(D\), and let \(\rho\) be an
update strategy for \(\Gamma\). \(\rho\) is symmetric if whenever \(s_1 \in LDB(D)\) and \(t_2 \in
LDB(V)\) are such that \(\rho(s_1, t_2)\) is defined, then \(\rho(\rho(s_1, t_2), \gamma'(s_1))\) is also
defined.

REQUIREMENT 4 An update strategy must be state independent.

EXAMPLE 1.2.12 Consider again the framework and initial states of Example
1.2.5. Suppose we stipulate that all updates to \(\Gamma_1\) be effected without
altering the state of \(\Gamma_2\). Suppose that we now want to delete \((s_2, p_2)\) from
the state of view $\Gamma_1$. This is impossible without deleting $(p_2, j_1)$ from the state of $\Gamma_2$. (We are not allowing null placeholders in this example, so it is not allowable to change $(s_2, p_2, j_1)$ to $(null, p_2, j_2)$; we will address examples later where this is allowed.) Note, however, that if the instance of $R_{SPJ}$ were instead

$$
R_{SPJ}:
$$

<table>
<thead>
<tr>
<th>S</th>
<th>P</th>
<th>J</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1$</td>
<td>$P_1$</td>
<td>$j_1$</td>
</tr>
<tr>
<td>$s_1$</td>
<td>$P_1$</td>
<td>$j_2$</td>
</tr>
<tr>
<td>$s_2$</td>
<td>$P_2$</td>
<td>$j_1$</td>
</tr>
<tr>
<td>$s_1$</td>
<td>$P_2$</td>
<td>$j_1$</td>
</tr>
</tbody>
</table>

then the deletion could be performed without changing the state of $\Gamma_2$, by deleting $(s_2, p_2, j_1)$, as $(p_2, j_1)$ would remain in the state of the relation of $\Gamma_2$ by virtue of the tuple $(s_1, p_2, j_1)$. Hence whether or not an update on a view is allowed depends upon information not visible in the view. This is an undesirable feature, which we rule out.

1.2.13 DEFINITION Let $D$ be a database schema, let $\Gamma = (V, \gamma)$ be a view of $D$, and let $\rho$ be an update strategy for $\Gamma$. $\rho$ is state independent if whenever $\rho(s_1, t_2)$ is defined, then so too is $\rho(r_1, t_2)$ for all $r_1$ for which $\gamma'(r_1) = \gamma'(s_1)$.

Our reference concept of a good admissible encompasses the requirements which have been stated above.

1.2.14 DEFINITION An update strategy $\rho$ is admissible if it is nonextraneous, functorial, symmetric, and state independent.
1.3 The Constant Complement Strategy

In this section, we review the constant complement update strategy of Bancilhon and Spyropoulos [Banc79], [BaSp81a], and [BaSp81b], with emphasis on ideas pertinent to our framework.

1.3.1 DEFINITION Let \( D \) be a database schema, and let \( \Gamma_1 = (V_1, \gamma_1) \) and \( \Gamma_2 = (V_2, \gamma_2) \) be views of \( D \).

(a) \( \Gamma_2 \) is a join complement of \( \Gamma_1 \) (or \( \Gamma_1 \) and \( \Gamma_2 \) are join complementary) if the mapping \( \gamma_1 \times \gamma_2 : \text{LDB}(D) \to \text{LDB}(V_1) \times \text{LDB}(V_2) \) defined by \( s \mapsto (\gamma_1'(s), \gamma_2'(s)) \) is injective. (Bancilhon and Spyropoulos call this just a complement; we append the adjective join for reasons which will become apparent later.)

(b) Let \( u = (s_1, (t_1, t_2)) \) be an update specification for \( \Gamma_1 \), and let \( s_2 \) be a solution for \( u \). The solution \( s_2 \) to \( u \) is performed with \( \Gamma_2 \) constant if \( \gamma_2'(s_1) = \gamma_2'(s_2) \).

(c) Let \( \rho \) be an update strategy for \( \Gamma_1 \). \( \rho \) is constant on \( \Gamma_2 \) if, for all \( s \in \text{LDB}(D) \) and \( t \in \text{LDB}(V_1) \) such that \( \rho(s, t) \) is defined, \( \gamma_2'(\rho(s, t)) = \gamma_2'(s) \).

The following theorem, whose proof is immediate from the injectivity of \( \gamma_1 \times \gamma_2 \), is the reason for considering updates relative to a complement.

1.3.2 THEOREM Let \( D \) be a database schema, and let \( \Gamma_i = (V_i, \gamma_i) \), \( i = 1, 2 \) be views of \( D \). Let \( u = (s_1, (t_1, t_2)) \) be an update specification to view \( \Gamma_1 \). If \( \Gamma_2 \) is a join complement of \( \Gamma_1 \), then there is at most one solution to to \( u \) which leaves \( \Gamma_2 \) constant. \( \square \)
We can thus solve the problem of deciding which of the many available solutions to an update specification to select by fixing a join complement of the view schema. Furthermore, an update strategy may always be extended to a relatively good one, as the next result shows.

1.3.3 PROPOSITION Let $D$ be a database schema and let $\Gamma_i = (V_i, Y_i), i = 1, 2$ be views of $D$ which are join complementary. Let $\rho$ be an update strategy for $\Gamma_1$ which is constant on $\Gamma_2$. Then $\rho$ can be extended to an update strategy $\hat{\rho}$ which is functorial and symmetric.

The problem now is, of course, how to choose an appropriate join complement. Clearly, the identity view $1 = (D, I)$ is a join complement to all views, and no updates at all (other than the identity) can be performed with $1$ constant. To avoid such a ludicrous anomaly, Bancilhon and Spyropoulos suggest using a minimal complement; that is, a complement which is "smaller" than any other complement. (We shall make the notion of smaller precise in the next section.) However, they then proceed to show that if one such minimal complement exists, then infinitely many do! Now they allow any function as a database mapping, and we might conjecture that the situation is more pleasant if we restrict attention to mappings defined by more reasonable operations. This is not the case, as we now show.

1.3.4 DEFINITION Let $D$ be a database schema, and let $\Gamma_i = (V_i, Y_i), i = 1, 2$ be views of $D$.

(a) $\Gamma_1$ and $\Gamma_2$ are meet complementary if the mapping $Y_1 \times Y_2 : \text{LDB}(D) \rightarrow \text{LDB}(V_1) \times \text{LDB}(V_2)$ is surjective.

(b) $\Gamma_1$ and $\Gamma_2$ are complementary if they are both join complementary and meet
complementary.

1.3.5 OBSERVATION Let $D$ be a database schema, and let $\Gamma_i = (V_i, \gamma_i)$, $i = 1, 2$ be views of $D$. If $\Gamma_1$ and $\Gamma_2$ are complementary, then any update to $\Gamma_1$ is possible with constant complement $\Gamma_2$. Hence any such update strategy is state independent.

Proof: By the above definition, $\gamma_1 \times \gamma_2': \text{LDB}(D) \rightarrow \text{LDB}(V_1) \times \text{LDB}(V_2)$ is bijective. Hence we may change the value of the first component to any legal value without altering the second. \[\blacksquare\]

There are two issues which must be resolved with this approach. First, a complement may not exist. Second, complements may not be unique. Example 1.2.5 provides an example of the former situation, while the next example illustrates the latter.

1.3.6 EXAMPLE Let $D$ be the schema with two unary relation symbols $R$ and $S$, and no constraints whatever. Let $\Gamma_1 = (V_1, \gamma_1)$ be the schema with the single relation symbol $R$, with $\gamma_1'$ the mapping which forgets $S$ and retains the value of the instance of $R$. Define $\Gamma_2$ similarly as the view which retains $S$ and forgets $R$. Define $\Gamma_3 = (V_3, \gamma_3)$ as the schema with single unary relation symbol $T$, with an element in the instance of $T$ iff it is either in the instance of $R$, or in the instance of $S$, but not both. An example set of instances is depicted below.
It is straightforward to verify that any two of these views are complementary (both join and meet), and hence none has a unique minimal complement. Still, $\Gamma_2$ seems like a much "nicer" complement of $\Gamma_1$ than does $\Gamma_3$ for at least two reasons. First of all, it is directly "part" of the base schema, obtained in fact as a sub-schema. Secondly, it allows for more natural update operations. For example, suppose we wish to insert $a_4$ into the instance of $R$ of the view $\Gamma_1$. With constant complement $\Gamma_2$, we simply insert $a_4$ into the instance of $R$ in the schema $D$, and we are done. With constant complement $\Gamma_3$, we must keep the instance of $T$ constant, which requires not only inserting $a_4$ into the instance of $R$ in the main schema, but also deleting $a_4$ from the instance of $S$. The former update is minimal, while the latter is not even nonextraneous.

It is the thesis of this work that there are formalizable properties which make the views $\Gamma_1$ and $\Gamma_2$ in the above example "nicer" than the view $\Gamma_3$. In the terminology of the next section, $\Gamma_1$ and $\Gamma_2$ are strong views of $D$, while $\Gamma_3$ is not. We shall propose that updates only be performed with respect to join complements which are strong views, and then only under certain further (reasonable) conditions.
2. **THE DECOMPOSITION FRAMEWORK**

The details of the update theory presented in this paper are based heavily upon the decomposition framework developed in [Hegn83] and [Hegn84]. In this section, the only the most essential features are outlined. For details, including all proofs, the reader is referred in particular to [Hegn84]. We presume a basic familiarity with the language of modern algebra [MaEl67], lattice theory [Grät78], and first-order logic [Ende72]. Also presumed is a basic familiarity with the ways in which first-order logic is used in database modelling (e.g., [Jaco82] or [Hegn84]).

2.1 **Schemata, Instances, Morphisms, and Views**

A type algebra is a triple $\Omega = (T,K,A)$, where:

(a) $T$ is a finite set of unary predicate symbols, called the types, which form a Boolean algebra under the logical operations of disjunction ($\lor$), conjunction ($\land$), and negation ($\neg$). The greatest element of the algebra is the universally true predicate $\tau_u$, while the least element is the universally false predicate $\tau_\bot$.

(b) $K$ is an at most countable set of constant symbols, called the names.

(c) $A$ is a set of axioms in the first-order language with equality whose (other) relation symbols are precisely those of $T$, and whose constant symbols are precisely those of $K$. The axioms are strong enough to determine, for each $k \in K$ and each $\tau \in T$, whether or not $\tau(k)$ is true.

Let $\Omega = (T,K,A)$ be a type algebra. A type assignment for $\Omega$ is any
model $\mu$ of $\mathbf{A}$. In other words, a type assignment for $\Omega$ is a set $X$, together with an assignment to each $\tau \in T$ of a set (= unary relation) $\text{Dom}_\mu(\tau)$, and an assignment of an element $k^\mu \in X$ to each $k \in K$, in a manner consistent with the constraints of $\mathbf{A}$.

The type algebra in our framework plays a role which extends that of attributes in the more traditional framework [Maie83]. In fact, we may incorporate attributes in our framework by assuming that for each attribute $A$, there is a type $\tau_A \in T$ which defines it. The advantage is that types may interact, whereas attributes do not. For example, if we wish attribute $C$ to be the union of attributes $A$ and $B$, the axiom $(\forall x)(\tau_C(x) \leftrightarrow \tau_A(x) \lor \tau_B(x))$ may be used to express this. Such interactions are highly useful in defining horizontal decompositions. Furthermore, we formalize null values as types with exactly one value (usually assigned a constant symbol). For example, $\tau_\eta(\eta) \land (\forall x)(\tau_\eta(x) \rightarrow x=\eta)$ defines the type $\tau_\eta$ to be the null-value type with null symbol $\eta$. We sometimes call $\eta$ the null value. These are "value inapplicable" nulls; we do not deal with incomplete information "value unknown nulls" in this work.

The type assignment plays a role similar to the attribute domains of the traditional framework. Within a given situation, the type assignment $\mu$ is fixed; in particular, a user is never allowed to update it. However, to make full use of the results of first-order logic, any type assignment which satisfies the axioms of the type algebra must be allowed.

The rest of our framework is very similar to (the relational part) of database logic of Jacobs [Jaco82]. A relational schema (over $\Omega$) is a pair $\mathbf{D} = (\text{Rel}(\mathbf{D}), \text{Con}(\mathbf{D}))$, where $\text{Rel}(\mathbf{D})$ is a finite set of relation symbols, and $\text{Con}(\mathbf{D})$ is a set of first-order constraints. In general, the elements of $\text{Con}(\mathbf{D})$ are in the language which includes the symbols of $\text{Rel}(\mathbf{D})$ and $\Omega$, as well
as equality. A (legal) database instance \( M \) for the schema \( D \) is just a model for both \( \text{Con}(D) \) and the type axioms \( A \). If the corresponding submodel of the type axioms is the type assignment \( \mu \), then we say that \( M \) is with respect to \( \mu \). The set of all legal instances of \( D \) with respect to \( \mu \) is denoted by \( \text{LDB}(D,\mu) \), while the class of all legal instances is denoted by \( \text{LDB}(D) \). Because we need to use results from model theory such as Beth's theorem, we do not restrict attention to finite models.

A database mapping between schemata \( D_1 \) and \( D_2 \) is given by an interpretation \( f \) of the language of \( D_2 \) into the language of \( D_1 \), which is the identity on the symbols of the type algebra \( \Omega \). Such a morphism is denoted by \( f : D_1 \rightarrow D_2 \). This interpretation defines a function \( f' : \text{LDB}(D_1) \rightarrow \text{LDB}(D_2) \) in the usual fashion. Note that by definition, \( f' \) preserves the underlying type assignment identically, so that we also have an induced mapping \( f'_\mu : \text{LDB}(D_1,\mu) \rightarrow \text{LDB}(D_2,\mu) \) for each type assignment \( \mu \).

A view of the schema \( D \) is a pair \( \Gamma = (V, \gamma) \), with \( V \) a schema (over the same type algebra \( \Omega \)), and \( \gamma : D \rightarrow V \) a database mapping with the property that the induced mapping \( \gamma' : \text{LDB}(D) \rightarrow \text{LDB}(V) \) is surjective. \( \text{View}(D) \) denotes the class of all views of the schema \( D \).

Before going on, we present an example which illustrates these ideas, and which will be used in later update examples.

2.1.1 EXAMPLE. In this example, we show how our framework allows the formalization of null values to permit join dependencies to define exact decompositions. The schema \( D \) is to have a single 4-ary relation symbol \( R \), with "attributes" \( \{A,B,C,D\} \), and with the join dependency constraint \( M[AB,BC,CD] \). We seek to make the join dependency exact in the sense that nulls will fill in for missing components. Start with four basic types \( \tau_A, \tau_B, \tau_C, \) and \( \tau_D \).
together with a null type $\tau_n$. The set of types of the type algebra is the free Boolean algebra generated by these five types. $n$ is the sole constant symbol. For convenience, let $\tau_A$ denote the type $\tau_A \lor \tau_n$; define $\tau_B$, $\tau_C$, and $\tau_D$ similarly. The overall "attribute" definition of the relation is then $R[A,B,C,D]$, which is expressed as a logical axiom by

$$(\forall x)(\forall y)(\forall z)(\forall w)(R(x,y,z,w) \implies \tau_A(x) \land \tau_B(y) \land \tau_C(z) \land \tau_D(w)).$$

We impose the further constraints that subsumed tuples (those with nulls) are present. More precisely, if a tuple of the form $(a,b,c,d)$ is present, then so too are $(a,b,c,n)$ and $(n,b,c,d)$. If a tuple of the form $(a,b,c,n)$ is present, so too are $(a,b,n,n)$ and $(n,b,c,n)$. If a tuple of the form $(n,b,c,d)$ is present, so too are $(n,n,c,d)$ and $(n,b,c,n)$. It is straightforward to express these rules as first-order constraints (in Con($D$)); see [Hegn84] for details.

The idea of allowing such tuples with nulls in the following manner is not new conceptually; it is the cornerstone of Sciore's notion of object [Scio80]. What is different about our presentation is that the rules defining their use are expressible directly in the language of constraints of the schema; there is no need to be informal or extralogical about the use of nulls. This allows us to make deductions of new constraints, as well as determine implied constraints on views, completely within the first-order language we have developed.

Finally, we have the join dependency $M[AB,BC,CD]$ expressed as it is intended for decomposition; namely, if tuples of the form $(a,b,n,n)$, $(n,b,c,n)$, and $(n,n,c,d)$ are present, then so too is a tuple of the form $(a,b,c,d)$. We also have the two embedded join dependencies $M[AB,BC]$ and $M[BC,CD]$: if tuples of the form $(a,b,n,n)$ and $(n,b,c,n)$ are present, so too is $(a,b,c,n)$; $M[BC,CD]$ is expressed similarly. All are easily converted to first-order sentences.
We have given the maximal representation for the join, in the terminology of [Scio80]. It is straightforward to re-axiomatize to remove subsumed tuples; see [Hegn84] for details. An example instance is given below.

\[
\begin{array}{cccc}
A & B & C & D \\
\hline
a_1 & b_1 & c_1 & d_1 \\
a_1 & b_1 & c_1 & n \\
a_1 & b_1 & n & n \\
n & b_1 & c_1 & d_1 \\
n & n & c_1 & d_1 \\
n & b_1 & c_1 & n \\
a_2 & b_2 & n & n \\
a_2 & b_3 & c_3 & n \\
a_2 & b_3 & n & n \\
n & b_3 & c_3 & n \\
n & n & c_4 & d_4 \\
\end{array}
\]

It is particularly convenient to define the component views from this schema. We define the views corresponding to the projections \( \pi_{AB}^\circ, \pi_{BC}^\circ, \pi_{CD}^\circ, \pi_{ABC}^\circ \) and \( \pi_{BCD}^\circ \) as usual, with the "\( \circ \)" superscript denoting that we only project the tuples with non-null values in at least two of the projected columns. Denote the corresponding views as \( \Gamma_{AB}^\circ = (V_{AB}^\circ, \pi_{AB}^\circ) \), etc. Note that the 3-column views inherit a join dependency constraint, while the two column views inherit only type constraints on their columns. Note in particular that \( \Gamma_{AB}^\circ \) and \( \Gamma_{BCD}^\circ \) (for example) are truly independent (i.e., they are meet complements); this is not so for the AB and BCD projections of an ordinary relation \( R \) satisfying \( \Pi(AB,BCD) \) because the common column B must match. We must formalize the null values to get this independence.

2.2 The Partial Lattice of Views

Let \( \Gamma_1 = (V_1, \gamma_1) \) and \( \Gamma_2 = (V_2, \gamma_2) \) be views of the same schema \( D \). A
morphism $f : \Gamma_1 \rightarrow \Gamma_2$ is a database mapping $f : V_1 \rightarrow V_2$ such that the following diagram commutes.

\[
\begin{array}{ccc}
\text{D} \\
\gamma_1 & \searrow & \gamma_2 \\
\downarrow & & \downarrow \\
V_1 & \longrightarrow & V_2
\end{array}
\]

$f$ is an isomorphism if there is another morphism $g : \Gamma_2 \rightarrow \Gamma_1$ such that $f \circ g$ and $g \circ f$ are each identities. $[\text{View}(D)]$ denotes the set of equivalence classes of isomorphic views of $\text{View}(D)$. We note that while $\text{View}(D)$ is usually a proper class, $[\text{View}(D)]$ is always a (countable) set.

2.2.1 PROPOSITION Let $\Gamma_1$ and $\Gamma_2$ be views of the same schema $D$.

(a) There is at most one morphism $f : \Gamma_1 \rightarrow \Gamma_2$.

(b) $\Gamma_1$ and $\Gamma_2$ are isomorphic if and only if there are morphisms $f : \Gamma_1 \rightarrow \Gamma_2$ and $g : \Gamma_2 \rightarrow \Gamma_1$. $\Box$

We say that $\Gamma_1 = (V_1, \gamma_1)$ implicitly defines $\Gamma_2 = (V_2, \gamma_2')$ if there is a function $h : \text{LDB}(V_1) \rightarrow \text{LDB}(V_2)$ such that $\gamma_2' = f \circ \gamma_1$. $\Gamma_1$ explicitly defines $\Gamma_2$ if there is a database morphism $f : \Gamma_1 \rightarrow \Gamma_2$. Clearly, if $\Gamma_1$ explicitly defines $\Gamma_2$, then it implicitly defines it (just take $h$ to be $f'$). What is remarkable is that the converse holds (thanks to Beth's theorem).

2.2.2 THEOREM Let $\Gamma_1$ and $\Gamma_2$ be views over the same schema $D$. Then $\Gamma_1$ explicitly defines $\Gamma_2$ if and only if it implicitly defines it. $\Box$.

Let $\text{Part}(\text{LDB}(D))$ denote the class of all partitions on $\text{LDB}(D)$ which only identify models based upon the same type assignment. It is easy to see that
Part(LDB(D)) forms a (big) lattice with universal bounds. The greatest element is the finest partition on LDB(D), while the least element is the partition which identifies, for each type assignment \( \nu \), all of the models with respect to \( \nu \). Join and meet are just partition supremum and infimum (see [Grät78, Ch. IV, §4]).

With each view \( \Gamma = (V, \gamma) \) of \( D \) we may associate an element \( \Pi(\Gamma) \) of Part(LDB(D)), namely, the kernel \( \ker(\gamma') \) of the mapping \( \gamma' : LDB(D) \rightarrow LDB(V) \). (Recall that the kernel of \( f : A \rightarrow B \) is the equivalence relation on \( A \) defined by \( x \equiv y \iff f(x) = f(y) \).) It is clear that \( \Gamma_1 \) implicitly defines \( \Gamma_2 \) iff \( \Pi(\Gamma_1) \) is finer than \( \Pi(\Gamma_2) \). Write \( \Gamma_2 \preceq \Gamma_1 \) iff \( \Gamma_1 \) defines (explicitly or implicitly) \( \Gamma_2 \). In effect, we have embedded \([\text{View}(D)]\) into the lattice Part(LDB(D)). Hence \( \preceq \) is a partial ordering on \([\text{View}(D)]\). Note that this embedding is nowhere near surjective. While Part(LDB(D)) is a proper class and not a set, in general, \([\text{View}(D)]\) is a countable set. Hence there are lots of "holes" in the embedding. In other words, very few of the partitions of LDB(D) arise as the result of database mappings. It is primarily on this issue that we differ with the modelling philosophy of Bancilhon and Spyritos [Banc79], [BaSp81a], [BaSp81b]. While they consider any partition to legitimately define a view, we only consider those (by comparison, very few) which arise as logical interpretations.

Joins and meets of views are defined as in Part(LDB(D)). It is not the case that joins and meets of views always exist as views; in other words, \([\text{View}(D)]\) does not form a sublattice of Part(LDB(D)), but only a \textit{relative (partial) sublattice} [Grät78, p. 40]. Note, however, that the finest partition corresponds to the identity view \( 1_D \), and the coarsest partition corresponds to the zero view \( 0_D \), and so we may speak of complements within \([\text{View}(D)]\). (The zero view preserves the type assignment \( \nu \), but contains no
other relations at all.) Referring back to our terminology of Definitions 1.3.1 and 1.3.4, it is easy to see that \( \Gamma_2 \) is a join complement of \( \Gamma_1 \) if \( \Gamma_1 \lor \Gamma_2 = 1_D \), and that it is a meet complement if \( \Gamma_1 \land \Gamma_2 = 0_D \). A complementary view then just reduces to a complement in the partial lattice \( \text{View}(D) \). Example 1.3.6 shows that complements in this partial lattice need not be unique.

2.3 Strong Views and their Relatives

Example 1.3.6 shows that some views seem to be better behaved than others. The formalization of strong views and strongly complemented strong views is an attempt to formalize this behavior. The concepts have a strong order-theoretic flavor, and we first state them independently of database concepts.

A \( \downarrow \)-poset is a partially ordered set \( P = (P, \preceq) \) with a least element \( \bot \). A mapping \( f : P \to Q \) between \( \downarrow \)-posets \( P = (P, \preceq) \) and \( Q = (Q, \preceq) \) is a morphism if it is monotonic and \( f(\bot) = \bot \). \( <P \to Q> \) denotes the set of all morphisms from \( P \) to \( Q \). \( <P \to Q> \) clearly forms a \( \downarrow \)-poset under the ordering \( f \preceq g \iff f(x) \preceq g(x) \) for all \( x \in P \). The least element is the function \( \bot \) which sends each \( x \) to \( \bot \).

Under the relation-by-relation inclusion of 1.2.3, \( \text{LDB}(D, \mu) \) forms a poset for each type assignment \( \mu \). We say that \( D \) has the null model property if the empty model (i.e., the model in which the instance of each \( R \in \text{Rel}(D) \) is empty) is in \( \text{LDB}(D, \mu) \), for each type assignment \( \mu \). If \( D \) has the null model property, then this null model is the least element in the ordering of \( \text{LDB}(D, \mu) \); hence it is a \( \downarrow \)-poset. (Actually, it suffices for each \( \text{LDB}(D, \mu) \) to
contain a minimal model which is definable by a single first-order sentence.)

The idea here is to exploit the algebraic properties of a view mapping $\gamma$ to ensure that it has nice properties. We first characterize these properties in terms of $\iota$-posets.

Let $P = (P, \iota)$ and $Q = (Q, \iota)$ be $\iota$-posets, and let $f \in \langle P \rightarrow Q \rangle$. $f$ admits least preimages if for any $y \in f(P)$, there is a least $x \in P$ such that $f(x) = y$. Denote the least preimage of $y$ by $y_f$. $lp(f)$ denotes the set of all $x \in P$ with least preimages. $f$ is least right invertible if it is surjective, admits least preimages, and the function $f^\# : Q \rightarrow P$ $y \mapsto y_f$ is in $\langle Q \rightarrow P \rangle$. The morphism $f^\# f \in \langle P \rightarrow P \rangle$ is called the endomorphism of $f$, and is denoted $f^\Theta$. $f$ is downward stationary if whenever $x \in lp(f)$ and $y \preceq x$, then $y \in lp(f)$ also. $f$ is a strong morphism (of $\iota$-posets) if it is downward stationary and least right invertible. $\langle P \rightarrow Q \rangle$ denotes the set of all strong morphisms from $P$ to $Q$.

Let $P = (P, \iota)$ again be a $\iota$-poset. A morphism $f \in \langle P \rightarrow P \rangle$ is called a strong endomorphism if it is idempotent (i.e. $f = f \circ f$) and downward stationary. The set of all strong endomorphisms on $P$ is denoted $\langle \langle P \rightarrow P \rangle \rangle$.

The relationship between strong morphisms and strong endomorphisms is a close one, as the following result shows.

2.3.1 LEMMA. Let $P = (P, \iota)$ and $Q = (Q, \iota)$ be $\iota$-posets.

(a) If $f \in \langle P \rightarrow Q \rangle$, then $f^\Theta \in \langle \langle P \rightarrow P \rangle \rangle$.

(b) If $g \in \langle \langle P \rightarrow P \rangle \rangle$, then the mapping $f^\# : P \rightarrow f(P)$ is in $\langle P \rightarrow Q \rangle$.

(Here $f(P)$ is the underlying set of the sub-$\iota$-poset $f(P) = (f(P), \iota)$ of $P$.)

It is easy to see that $\langle \langle P \rightarrow P \rangle \rangle$ forms a partially ordered set with universal bounds under the ordering and least element inherited from $\langle P \rightarrow P \rangle$;
the greatest element is the identity function \(1_P\). What is more significant is the following.

2.3.2 LEMMA Let \(P\) be a \(1\)-poset.

(a) Complements are unique in \(<<P \rightarrow P>>\), and the complemented elements form a Boolean algebra.

(b) If \(f\) and \(g\) are complements in \(<<P \rightarrow P>>\), then the mapping \(f \times g : P \rightarrow f(P) \times g(P)\) is a \(1\)-poset isomorphism. (Here \(f(P) \times g(P) = (f(P) \times g(P), \leq)\) is a \(1\)-poset under the obvious operations.) D

These notions translate directly to database mappings and views. The database mapping \(f : D_1 \rightarrow D_2\) is strong if \(f'_\mu : \text{LDB}(D_1, \mu) \rightarrow \text{LDB}(D_2, \mu)\) is a strong morphism (of \(1\)-posets) for each type assignment \(\mu\). The view \(\Gamma = (V, \gamma)\) is strong if \(\gamma\) is a strong database mapping. Similarly, a database mapping \(f : D_1 \rightarrow D_1\) is a strong endomorphism if \(f'_\mu\) is a strong endomorphism (of \(1\)-posets) for each type assignment \(\mu\). A strong view of the schema \(D\) is a view \(\Gamma = (V, \gamma)\) for which \(\gamma\) is a strong morphism. \(\Gamma\) is strongly complemented if it is a strong view with a complement which is also a strong view.

The superscripts \(\emptyset\), \#, and \$ are used on database mappings with their same meaning as in \(1\)-poset morphisms. For example, if \(f : D_1 \rightarrow D_2\) is a strong database mapping, then \(f^\# : D_2 \rightarrow D_1\) denotes its least right inverse, defined as the unique database mapping given for each type assignment \(\mu\) by the least right inverse of \(f'_\mu\).

2.3.3 THEOREM Let \(D\) be a database schema with the null model property, and let \(\Gamma_1 = (V_1, \gamma_1)\) and \(\Gamma_2 = (V_2, \gamma_2)\) be strong views of \(D\).
(a) $\Gamma_1 \leq \Gamma_2$ (as elements of $\text{View}(D)$) iff for each type assignment $\nu$, $(\gamma_1)_{\nu}$ $\leq (\gamma_2)_{\nu}$ (as elements of the $\nu$-poset $\text{LDB}(D,\nu) \rightarrow \text{LDB}(D,\nu)$).

(b) If $(\gamma_1)_{\nu}$ is a complement of $(\gamma_2)_{\nu}$ in $\text{LDB}(D) \rightarrow \text{LDB}(D)$ for each type assignment $\nu$, then $\Gamma_1$ is a complement of $\Gamma_2$ in $\text{View}(D)$. Furthermore, $\Gamma_2$ can have at most one complement of this form. Thus, strong complements are unique.

It follows from 2.3.2(a) and 2.3.3(b) that the strongly complemented strong views form a Boolean algebra, as a subalgebra of $\text{View}(D)$. We call this algebra the component algebra of $D$. The strong complement of $\Gamma = (V,\gamma)$ is denoted $\Gamma^C = (V^C,\gamma^C)$. It also follows from 2.3.1 that a strong view $\Gamma = (V,\gamma)$ is entirely determined by its endomorphism $\gamma^\emptyset$. In fact, if the schema $D$ is endowed with all of the implied constraints of the mapping $\gamma^\emptyset : D \rightarrow D$, yielding $D^\emptyset$, we get a view $\Gamma^\emptyset = (D^\emptyset,\gamma^\emptyset)$ which is isomorphic to $\Gamma$, but which has the same relation symbols as the base schema. An example will illustrate.

2.3.4 EXAMPLE Return to the setting of Example 2.1.1. Each of the $\Gamma^\emptyset$ views defined there is strong. Let us consider specifically $\Gamma_{AB}^\emptyset$. Monotonicity and $\iota$ preservation are obvious. For any relation instance of the view, the least preimage is the base instance which appends nulls to the last two columns. An example is given below.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a_1$</td>
<td>$b_1$</td>
<td>$n$</td>
<td>$n$</td>
</tr>
<tr>
<td>$a_2$</td>
<td>$b_2$</td>
<td>$n$</td>
<td>$n$</td>
</tr>
</tbody>
</table>

View instance Base Schema Instance

The least right inverse $(\pi_{AB}^\emptyset)^\#$ is defined by this association. Downward
stationarity is obvious, as any base relation for which all tuples have nulls in the last two columns is a least preimage.

The database mapping \((r_{AB}^\circ)^{\oplus}_D : \text{LDB}(D) \rightarrow \text{LDB}(D)\) is just the restriction of the base relation to tuples with nulls in the last two columns. Formally, if \(R_b\) denotes the relation symbol of the base schema, and \(R_v\) the relation symbol of the view, then the defining formula is

\[(\forall x)(\forall y)(\forall z)(\forall w)(R_v(x,y,z,w) \leftrightarrow R_b(x,y,z,w) \land r_{AB}(x) \land r_B(y) \land r_\eta(z) \land r_\eta(w)) \]

We call such a view a restriction or object; the defining database mapping is denoted \(\rho(R(r_{AB},r_B,r_\eta,r_\eta))\). Such a mapping is considered to be a generalization of Sciore's notion of object \([\text{SciO80}]\). It is precisely this idea that the view (= "quotient mapping") may be regarded as a subschema (= "immersion") that characterizes strong views. (The mathematical generalization here is that of a component of a direct sum; if \(X\) is some algebraic object and \(X = X_1 \oplus X_2\), then we have both a projection \(X \rightarrow X_1\) and an injection \(X_1 \rightarrow X\). The same idea is working here; we think of the base relation \(R(ABCD)\) as being the "direct sum" (= join) of the components \(R(AB), R(BC),\) and \(R(CD)\). For a much more detailed discussion of the aspects of algebraic decomposition, see \([\text{Hegn84}]\).)

To get \((r_{AB}^\circ)^{\oplus}_D\), we enforce (on \(D^D\)) the constraint that the \(C\) and \(D\) columns contain only nulls.

The component algebra is generated by \(r_{AB}^\circ, r_{BC}^\circ, r_{CD}^\circ\). The other members are then \(1_D, 0_D, r_{ABC}^\circ, r_{BCD}^\circ,\) and \(r_{AB}^\circ r_{CD}^\circ\). The last view consists of two relations, one the \(AB\) projection and the other the \(CD\) projection. Its endomorphism projects down all tuples of the form \((a,b,n,n)\) and \((n,n,c,d)\).

The strong complement of \(r_{AB}^\circ\) is \(r_{ECD}^\circ\); this is easily verified. Other complements are computed similarly. However, we should remark that it is possible to produce examples of strong views which do not have strong comple-
ments.

A view which is isomorphic to a strong view (resp. strongly complemented strong view) is called a generalized strong view (resp. generalized strongly complemented strong view). Most of our subsequent results carry over to this more general case.
3. **VIEW UPDATING USING STRONG JOIN COMPLEMENTS**

All of the results of this section depend upon the condition that the base schema have the null model property. Therefore, for all of this section, D is assumed to be a relational database schema with the null model property. We also assume that an update leaves the type assignment fixed.

3.1 **Updating Strongly Complemented Strong Views.**

Updating a strongly complemented strong view is the cleanest situation, as updates are always possible, and are always admissible. As this situation forms the basis for the analysis of more complex cases, we consider it first. We know from section 1 that updates relative to complements are quite well behaved; what we get in this case is the added property of nonextraneousness.

3.1.1 **THEOREM** Let \( \Gamma_1 = (V_1, \gamma_1) \) be a strong view of D, and let \( \Gamma_2 = (V_2, \gamma_2) \) be a strong complement of D. Then any update specification \( S = (s_1, (t_1, t_2)) \) to \( \Gamma_1 \) has a unique solution with constant complement \( \Gamma_1 \). This solution is admissible.

Proof outline: By Theorem 1.3.2, the solution is unique, and by Proposition 1.3.3, functorial and symmetric. By Theorem 1.3.5, any update is possible. Hence, to show admissibility, all we need establish is that the solution is nonextraneous. By 2.3.4, there is a \( \mathcal{I} \)-poset isomorphism \( h = (\gamma_1^{\Theta})_u \times (\gamma_2^{\Theta})_u : \text{LDB}(D,u) \rightarrow (\gamma_1^{\Theta})_u(\text{LDB}(D,u)) \times (\gamma_2^{\Theta})_u(\text{LDB}(D,u)) \). Clearly, an update to \( \Gamma_1 \) with constant complement \( \Gamma_2 \) will only change the second component; that is, if \( h(s_1) = (t_1, s_2) \) and \( r \) is the unique solution with constant complement, then
h(r) = (t_2, s_2). Now suppose that p is a solution to S with p \Delta s \subseteq r \Delta s. Let (t_2, p_2) = h(p). Since \( Y_2^{\emptyset} \) is idempotent and downward stationary, p_2 \subseteq p, s_2 \subseteq r, and s_2 \subseteq s. Hence p_2 \subseteq s_2 (else p \Delta s \not\subseteq r \Delta s). Similarly, s_2 \subseteq p_2, so that s_2 = p_2. Hence r is a nonextraneous solution. □

3.2 Updating Arbitrary Views

Let \( \Gamma_1 \) be any view of D (not necessarily strong). A strongly complemented strong view \( \Gamma_2 \) is a strong join complement of \( \Gamma_1 \) if \( \Gamma_2^c \not< \Gamma_1 \).

3.2.1 LEMMA Let D be a database schema with the null model property, and let \( \Gamma_1 \) and \( \Gamma_2 \) be views of D, with \( \Gamma_2 \) strong.

(a) If \( \Gamma_2 \) is a strong join complement of \( \Gamma_1 \), then it is also a join complement (in the sense that \( \Gamma_1 \mid \Gamma_2 = 1_D \) in \( \text{View(D)} \)).

Proof: (a) \( \Gamma_2^c \not< \Gamma_1 \) means that \( \Pi(\Gamma_2^c) \) is coarser than \( \Pi(\Gamma_1) \), and since \( \sup(\Pi(\Gamma_2^c), \Pi(\Gamma_2)) \) is the identity partition, surely \( \sup(\Pi(\Gamma_1), \Pi(\Gamma_2)) \) is also the identity partition. □

3.2.2 MAIN UPDATE THEOREM Let \( \Gamma_1 \) be any view of D and let \( S = (s_1, (t_1, u_1)) \) be any update specification on \( \Gamma_1 \).

(a) Let \( \Gamma_2 \) be a strong join complement of \( \Gamma_1 \). If S has a solution with constant complement \( \Gamma_2 \), then this solution is admissible.

(b) If \( \Gamma_2 \) and \( \Gamma_3 \) are each strong join complements of \( \Gamma_1 \) for which S has a solution, then the solution is the same in each case. In other words, if an update specification is performed with respect to a join complement which is a strong view and whose complement is smaller than \( \Gamma_1 \), then the solution of the
update is independent of the particular complement chosen.

Proof outline: (a) Let \( s_2 \) be the unique solution to \( S \). Since \( \Gamma_2 \) is a strong join complement of \( \Gamma_1 \), \( \Gamma_2^C \triangleleft \Gamma_1 \), and so there is a surjective database morphism \( f : \Gamma_1 \rightarrow \Gamma_2^C \), by 2.2.2. Hence \( s_2 \) must also be a solution to the update specification \((s_1, (f'(t_1), f'(t_2)))\) which is guaranteed to be admissible by Theorem 3.1.1.

(b) Suppose that \( S \) has a solution for both constant complement \( \Gamma_2 \) and constant complement \( \Gamma_3 \). Denote the solutions to \( S \) for these complements by \( s_2 \) and \( s_3 \), respectively. First, note that \((\gamma_2 \triangleleft \gamma_3)'(s_2) = (\gamma_2 \triangleleft \gamma_3)'(s_3) = (\gamma_2 \triangleleft \gamma_3)'(s_1)\), just by definition of constant complement. Next, \((\gamma_2^C)'(s_2) = (\gamma_2^C)'(s_3)\), since \( \gamma_1'(s_2) = \gamma_1'(s_3) \), and \( \Gamma_2^C \triangleleft \Gamma_1 \). Similarly, \((\gamma_3^C)'(s_2) = (\gamma_3^C)'(s_3)\). Hence \((\gamma_2^C \triangleleft \gamma_3^C)'(s_2) = (\gamma_2^C \triangleleft \gamma_3^C)'(s_3)\). Thus, \( s_2 = s_3 \). \( \blacksquare \)

3.2.3 UPDATE PROCEDURE To service an update specification \((s_1, (t_1, t_2))\) to schema \( \Gamma_1 = (\mathbf{V}_1, \gamma_1) \) with strong complement \( \Gamma_2 = (\mathbf{V}_2, \gamma_2) \), proceed as follows. Let \( f : \Gamma_1 \rightarrow \Gamma_2^C \) be the unique view morphism, guaranteed to exist because \( \Gamma_2^C \triangleleft \Gamma_1 \). The first step is to determine the solution \( s_2 \) to the update specification \((s_1, (f'(t_1), f'(t_2)))\) to \( \Gamma_2^C \) which exists uniquely and nicely, by 3.1.1. Next, determine \( \gamma_1'(s_2) \). If this is the update requested (i.e., \( t_2 \)), we have succeeded. Otherwise, the update is not possible with constant complement \( \Gamma_2 \).

Note that the update procedure has much in common with the query modification strategy of Stonebraker [Ston75]. We illustrate further with an example.

3.2.4 EXAMPLE We use the framework of Example 2.1.1. Consider the view \( \Gamma_{ABD} \)

32
\[ \vartheta = (\mathcal{V}_{AB}, \pi_{AB}) \], which just projects the three columns of the base relation, with no regard for types or nulls. Thus, the view state corresponding to the example state of 2.1.1 will be

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>a_1</td>
<td>b_1</td>
<td>d_1</td>
</tr>
<tr>
<td>a_1</td>
<td>b_1</td>
<td>n</td>
</tr>
<tr>
<td>n</td>
<td>b_1</td>
<td>d_1</td>
</tr>
<tr>
<td>n</td>
<td>n</td>
<td>d_1</td>
</tr>
<tr>
<td>n</td>
<td>b_1</td>
<td>n</td>
</tr>
<tr>
<td>a_2</td>
<td>b_2</td>
<td>n</td>
</tr>
<tr>
<td>a_2</td>
<td>b_3</td>
<td>n</td>
</tr>
<tr>
<td>n</td>
<td>b_3</td>
<td>n</td>
</tr>
<tr>
<td>n</td>
<td>n</td>
<td>d_4</td>
</tr>
</tbody>
</table>

The constraints may be summarized by stating that (i) for any tuple of the form \((a,b,d)\), there are also tuples of the form \((a,b,n)\) and \((n,b,d)\); (ii) for any tuple of the form \((n,b,d)\), there are also tuples of the form \((n,b,n)\) and \((n,n,d)\); (iii) there are no tuples of the form \((a,n,d)\), \((a,n,n)\), or \((n,n,n)\).

The smallest strong join complement of \(\Gamma\) is \(\Gamma_{BCD}^o\). Therefore all updates must be "filtered" through the complementary view \(\Gamma_{AB}^o\). In other words, the only updates which can be performed are those which could be achieved by updating the state of \(\Gamma_{AB}^o\). Let \(f : \Gamma \rightarrow \Gamma_{AB}^o\) be the unique view morphism, guaranteed because \(\Gamma_{AB}^o \preceq \Gamma\). \(f\) is easily seen to be the mapping which first projects the A and B columns (dropping D), and then saves only those tuples without \(n\) entries.

Suppose that the update request is to delete \((a_2,b_3,n)\) and \((n,b_3,n)\). This maps to deleting \((a,b)\) from \(\mathcal{V}_{AB}\), which achieves the desired effect, so that the update is allowed. On the other hand, suppose that the update request is to delete \((n,n,d_4)\). (Note that the resulting state of the view is legal.) This maps to doing nothing in \(\mathcal{V}_{AB}\); hence the update cannot be effected with constant complement \(\Gamma_2\).
3.3 Updating Strong Views

The need to update with respect to a strong join complement, and not just a join complement, is crucial, as the next example shows.

3.3.1 EXAMPLE Refer back to Example 1.3.6. It is clear that $\Gamma_1$ and $\Gamma_2$ are strongly complemented strong views and each other's complement in the component algebra. $\Gamma_3$ is also a complement of each, although it is not even a strong view. Suppose we update $\Gamma_1$ with constant complement $\Gamma_3$. Specifically, suppose insert $a_4$ into the state of $\Gamma_1$. As shown in Example 1.3.6, this produces an extraneous deletion of $a_4$ in $S$, violating the admissibility of the update.

The problem illustrated in the above example cannot occur if the view to be updated is a strong view. We defer the proof of the following result to a full paper, as it uses concepts from [Hegn04] which we have not discussed here.

3.3.1 LEMMA Let $\Gamma_1$ and $\Gamma_2$ be strong views of $D$, and assume further that $\Gamma_1$ is strongly complemented. Then for $\Gamma_1$ to be a strong join complement of $\Gamma_2$, it suffices that it be an ordinary join complement. □
REFERENCES


