Using Constraint Satisfaction for View Update Translation

Hua Shu[1]

Abstract. View update is the problem of translating update requests against a view into update requests against the base data. In this paper, we present a novel approach to this problem in relational databases. Using *conditional tables* to represent relational views, we translate a view update problem into a disjunction of a number of *constraint satisfaction problems* (CSPs). Solutions to the CSPs correspond to possible translations of the view update. Semantic information to resolve ambiguity can be handled as additional constraints in the CSPs. This approach enables us to apply the rich results of the CSP research to analyze and solve an important problem in database management.

1 Introduction

In a relational database, a view is a virtual relation schema that makes it possible for a user to focus on relevant data. Views are usually specified using database query languages. They may be *materialized* (i.e., a physical copy of a view is stored and maintained) or *virtual* (i.e., relevant information about the view is computed as needed). The view update problem refers to the problem of updating a database by issuing update requests against views. This well-known problem has attracted a lot of attention in the database community. Various procedures of view update translation into updates against the base relations are defined (e.g., [2, 3, 7, 11]). The problem is that, given a view update request, the corresponding updates of the base data may not be unique, not exist or be ill-defined. To determine the *unique* translation of a view update request that maintains consistency and is free of side effects, it is necessary to use additional information concerning the database or application semantics [2, 7, 9, 8, 12]. The general translation procedures proposed so far are mostly procedural and ad hoc, where problem formalization and problem solving are not separated. Adding new semantic information to resolve ambiguity may change the problem specification and render the procedures inapplicable.

Our contribution is to show that the problem of view update translation can be transformed into constraint satisfaction problems (CSPs) by means of conditional tables. Each solution to the CSPs corresponds to one possible translation of the view update. Semantic information to resolve ambiguity can be handled by adding constraints to the CSPs. The proposed approach enables us to utilize the rich results of the growing CSP research, including the large number of constraint satisfaction algorithms and techniques [14]. This paper is an extended abstract of [13], where more detail, including the proofs of the theorems, can be found.

This paper is organized as follows. First, we briefly describe the view update translation and the problem considered in this paper, as well as conditional tables. Then we use simple examples to illustrate the basic ideas of our approach. We consider the translations of view deletions and insertions only. View modifications can be handled as the compositions of deletions and insertions. We present the transformation more rigorously in Section 5. Finally, the complexity of this approach is discussed.

2 View Update Translation

Let I denote an instance of a database scheme. Usually there are a set of semantic constraints on the database scheme. Let's assume that I satisfy the constraints (i.e., I is *consistent*). Suppose that there is a view defined by some relational algebra expression q. The instance of the view with respect to I is denoted by $q(I)$. Updates are modeled as mappings from one instance of a database to another instance of the database. Let μ denote a request for an update (deletion or insertion) of a view. The updated view is denoted by $\mu(q(I))$. The problem is how to define a *translator* that translates μ into an update μ' against the database I so that the updated database $\mu'(I)$ is consistent and $q(\mu'(I)) = \mu(q(I))$ holds.

As an example, consider a view V, defined by a relational algebra expression $\pi_{A,C}(T_1 \bowtie T_2)$ over relations T_1 and T_2 shown in Table 1. $\pi_{A,C}$ stands for *projection* on attributes A and C. $\bowtie$ stands for *natural join* which requires that the tuples that are concatenated agree on the common attributes.

Table 1.

T_1

A	B
a	b
a'	b

T_2

B	C
b	c
b'	c'

$V : \pi_{A,C}(T_1 \bowtie T_2)$

A	C
a	c
a'	c

Let each of the attributes A, B and C be associated with a domain, denoted by $Dom()$. It is assumed that $Dom(A) = \{a, a'\}, Dom(B) = \{b, b'\}$ and $Dom(C) = \{c, c'\}$. Suppose that a user issues a request to delete tuple (a, c) from V. The effect of this deletion is a relation containing only one tuple

[1] Institut für Informatik, Universität Hannover, Lange Laube 22, D-30159 Hannover, Germany. Current address: Dept. of Computer Science, University of Karlstad, 65188 Karlstad, Sweden. Email: Hua.Shu@hks.se.

ECAI 98. *13th European Conference on Artificial Intelligence*
Edited by Henri Prade
Published in 1998 by John Wiley & Sons, Ltd.

(a', c). The question is how to translate this deletion request into update requests against the base relations T_1 and T_2.

Generally, a view deletion request may be translated into deletions and modifications. To determine a translation of the deletion request, it is important to identify tuples in T_1 and T_2 that give rise to, and only to, the tuple (a, c). Such tuples are called the "clean sources" of (a, c) [2]. The tuples (a, b) in T_1 and (b, c) in T_2 are said to be the "sources" of the tuple (a, c) in the view, since deleting either of the tuples would lead to the deletion of (a, c) from the view. However, deleting (b, c) also leads to the deletion of the tuple (a', c) from the view. Thus only the tuple (a, b) is a "clean source" of the tuple (a, c). It follows that an exact translation of the deletion request is the deletion of (a, b) from the relation T_1. It is worth noticing that the view-dependency approach [2] fails to find this translation.

Ambiguity is a major problem in view update translation: there are usually multiple possible translations of a view update, and there are usually any number of possible translations. These translations are a priori equivalent, but may be ranked with respect to a set of predefined preference criteria. Thus translating view updates usually involves the iteration of two steps: First check whether there is a unique translation using the given semantic information and find the translation(s) of the view update; Secondly, if there are multiple translations, ask the user for more information to resolve the ambiguity, or ask the user to choose one among the possible translations.

The problem considered in this paper is concerned with the first step: search for consistent states I' of the database that satisfy $q(I') = \mu(q(I))$. Apparently, any μ' satisfying $I' = \mu'(I)$ is a translation of μ. We show in this paper that this problem can be transformed into CSPs. Some of the semantic information imposed to resolve ambiguity can readily be handled as additional constraints. Some others can be used as preference policies to choose optimal translations. For instance, a requirement of *minimal updates* can be adopted to prefer a translation performing minimal changes to the database I. Another possible requirement is that the user should be able to cancel the effect of every update on a view, i.e., $\mu'(I) = I$ whenever $\mu(q(I)) = q(I)$ [1].

3 Conditional Tables

In the relational data model, a free tuple is a tuple containing variables. A *conditional table* (C-table) [6] is a tuple $T_c = (T, \phi)$, where T is a finite set of free tuples, and ϕ is a mapping over T that associates a logical formula $\phi(t)$ with each tuple t of T. $\phi(t)$ is called the *local condition* associated with t. The conditions are expressions built up by means of Boolean connectives $\neg, \wedge, \vee$ and $\Rightarrow$ from constants **true**, **false**, and *equality atoms* of the forms $x = y$ and $x = c$, where x and y are variables and c is a constant. The conditions are represented under a column labeled *con*. A *c-tuple* is a tuple t of T together with local condition $\phi(t)$. A C-table is called a *naive table* if the local conditions are **true**. A C-table is called a *Horn table* if the local conditions are conjunctions of atomic equalities.

Let η denote a special term representing an undetermined value. Similar to an attribute, a variable has a value domain. It is assumed that η is not contained in the domain of any attribute, but may be contained in the domain of a variable. η is used to model the deletion of a tuple in a relation.

We denote the evaluation of a condition C with respect to a valuation $\mathbf{v}$ by $V^{\mathbf{v}}(C)$. The evaluation of a condition is defined recursively as follows:

1. For an equality atom $x = y$, $V^{\mathbf{v}}(x = y) = \mathbf{U}$ if $\mathbf{v}(x) = \eta$ or $\mathbf{v}(y) = \eta$. $V^{\mathbf{v}}(x = y) = \mathbf{T}$ if $\mathbf{v}(x) \neq \eta$, $\mathbf{v}(y) \neq \eta$ and $\mathbf{v}(x) = \mathbf{v}(y)$. Otherwise, $V^{\mathbf{v}}(x = y) = \mathbf{F}$.
2. **true** is always evaluated $\mathbf{T}$ and **false** always $\mathbf{F}$.
3. The evaluation of conditions with Boolean connectives is based on the *strong Kleene logic* [10].

With respect to valuation $\mathbf{v}$, a condition C is said to be *satisfied* if $V^{\mathbf{v}}(C) = \mathbf{T}$, *falsified* if $V^{\mathbf{v}}(C) = \mathbf{F}$ and *undefined* if $V^{\mathbf{v}}(C) = \mathbf{U}$. We say that two conditions C_1 and C_2 are *equivalent* if and only if $V^{\mathbf{v}}(C_1) = V^{\mathbf{v}}(C_2)$ for any valuation $\mathbf{v}$. Conditions equivalent to **true** can be omitted in the c-tables. If all local conditions in a C-table are **true**, then the *con* column can be omitted.

The instance of a free tuple t with respect to a valuation $\mathbf{v}$ is denoted by $\mathbf{v}(t)$. If $\mathbf{v}$ assigns η to some variable in tuple t, then we say that $\mathbf{v}(t)$ is *non-existing*, denoted $\mathbf{v}(t) = \eta$. For a C-table $T_c = (T, \phi)$,

$$\mathbf{v}(T_c) = \{\mathbf{v}(t) \mid t \in T, V^{\mathbf{v}}(\phi(t)) = \mathbf{T} \text{ and } \mathbf{v}(t) \neq \eta\} \quad (1)$$

We call $\mathbf{v}(T_c)$ the *state* of T_c with respect to $\mathbf{v}$. For a set I_c of C-tables, $\mathbf{v}(I_c) = \{\mathbf{v}(T_c) \mid T_c \in I_c\}$. $rep(T_c)$ denotes the collection of all possible states of T_c.

We do not review the definitions of relational operators on C-tables. Interested readers may consult [6, 4] or [13]. Table 2 illustrates a C-table representing a relational view $\pi_{A,C}(T_1^V \bowtie T_2^V)$, where T_1^V and T_2^V are naive tables.

Table 2.

T_1^V

A	B
t	x
s	y

T_2^V

B	C
u	z
v	r

$\pi_{A,C}(T_1^V \bowtie T_2^V)$

A	C	con
t	z	$x = u$
s	z	$y = u$
t	r	$x = v$
s	r	$y = v$

Conditional table can be evaluated in a semantically correct manner with respect to the primitive relational operators (namely, selection, projection, Cartesian product, intersection, join, union and difference) [6]. Given a set I_c of C-tables and a query q, we denote the result of the query on the set I_c of C-tables by $q(I_c)$. It has been proven (as Theorem 7.1 in [6]) that for any PSUJ operator q (i.e., en operator defined using *projection*, *selection*, *union* and *join*) and any set I_c of C-tables,

$$\mathbf{v}(q(I_c)) = q(\mathbf{v}(I_c)) \quad (2)$$

holds for any valuation $\mathbf{v}$ of the variables in I_c. That implies, *for any PSUJ query q and any set I_c of C-tables, $q(I_c)$ can be represented by a C-table* [6]. Although the value η are not considered in [6], the above result still applies when η is introduced, because the proof of the result has only to do with the definitions of the relational operators. The definitions are not affected by the introduction of η. The time/space taken to evaluate the operations on a set of C-tables is bounded by a polynomial in the maximal size of the C-tables ([4], Theorem 5.1).

4 The Basic Ideas

In this work, we consider relational views defined in terms of relational operators projections, selections and joins involving only equalities. For simplicity, we consider only update operations that deal with single tuples in a relation.

A *binary CSP* is a CSP with constraints involving one or two variables. A CSP with constraints involving more variables are called a *general CSP*. In this context, the constraints considered are the *conditions* defined in Section 3. Research on CSPs is usually restricted to binary CSPs. It is because a general CSP can be transformed into a number of binary CSPs. If a constraint C in a general CSP is a disjunctive condition $C_1 \vee ... \vee C_n$, then the CSP can be transformed into n sub-CSPs, each being the result of replacing C in the original CSP with one of the disjuncts. If a constraint is a conjunctive condition $C_1 \wedge ... \wedge C_n$, then the connective $\wedge$ can simply be omitted. This eventually leads to a set of CSPs containing no disjunction. The resulting CSPs are binary CSPs because the constraints are equality atoms (of the form $x = y$ or $x = c$), or negations of the equality atoms. In this paper, we specify such binary CSPs as a tuple $(\Gamma, D, \mathcal{C}^+, \mathcal{C}^-)$, where Γ is a finite set of variables, D is a function which maps each variable x in Γ to the domain of x, $\mathcal{C}^+$ contains equality atoms to be satisfied and $\mathcal{C}^-$ contains equality atoms not to be satisfied. The constraints in $\mathcal{C}^+$ are said to be positive. The constraints in $\mathcal{C}^-$ are said to be negative. A *solution* to the constraint satisfaction problem is a valuation of the variables in Γ that satisfies all constraints in $\mathcal{C}^+$ and falsifies all constraints in $\mathcal{C}^-$. A CSP is said to be *solvable* if at least one solution exists. If a negative constraint is derivable from the positive constraints, or a positive constraint is derivable from the negative constraints, then the CSP is not solvable. Due to space limit, we do not review the relevant approaches to solving CSPs. Interested reader may consult the book by Tsang [14].

An attribute is said to be *distinguished* in a view defined by a relational expression E if it appears in the set of attributes involved in E. An attribute is said to be *exposed* if it is used in the selection condition in E [5]. The union of the distinguished and exposed attributes is called the *extended attribute set* of E [5]. Recall the relations shown in Table 1. Attributes A and C are distinguished, whereas attribute B is exposed. The extended attribute set of the view contains attributes A, B and C.

4.1 Translating deletion request

Recall the relations shown in Table 1. Consider the deletion of tuple (a, c) from V. Our approach to determining the translation is as follows. First of all, we replace the values in the relations T_1 and T_2 of all attributes in the extended attribute set with variables. The outcome is the tables T_1^V and T_2^V shown in Table 2. The valuation of the variables is

$$\mathbf{v}_1 = \{t/a, s/a', x/b, y/b, u/b, v/b', z/c, r/c'\}.$$

The domains of the variables, specified by function D_η, are the domains of the corresponding attributes extended with η, i.e., $D_\eta(t) = D_\eta(s) = Dom(A) \cup \{\eta\} = \{a, a', \eta\}$, $D_\eta(x) = D_\eta(y) = D_\eta(u) = D_\eta(v) = Dom(B) \cup \{\eta\} = \{b, b', \eta\}$, and $D_\eta(z) = D_\eta(r) = Dom(C) \cup \{\eta\} = \{c, c', \eta\}$. T_1^V and T_2^V are called the *DM-version* of the relations T_1 and T_2 with respect to the view; D stands here for deletion and M for modification. The conditional table shown in Table 2 represents the view $\pi_{A,C}(T_1^V \bowtie T_2^V)$.

Now we want to find out the constraints under which the effect of the deletion is a state of the C-table $\pi_{A,C}(T_1^V \bowtie T_2^V)$. For simplicity, we consider the case in which the valuations of t, s, z and r are defined as $\mathbf{v}_1$. Let

$$\mathbf{v}_1' = \{t/a, s/a', z/c, r/c'\}.$$

Since only evaluating the tuple (s, z) with respect to $\mathbf{v}_1$ results in tuple (a', c), the associated condition $y = u$ is the only one that should be satisfied; the other three conditions $x = u$, $x = v$ and $y = v$ should be falsified. We can construct the following binary CSP:

$$(\{x, y, u, v\}, D_\eta, \{y = u\}, \{x = u, x = v, y = v\}). \quad (3)$$

There are a number of solutions to this CSP. For example, one of them is $\mathbf{v}_2 = \{x/\eta, y/b, u/b, v/b'\}$. Recall that the valuations of t, s, z and r are defined as $\mathbf{v}_1$. $(\mathbf{v}_1' \cup \mathbf{v}_2)(T_1^V)$ and $(\mathbf{v}_1' \cup \mathbf{v}_2)(T_2^V)$ are the updated base relations. Comparing the updated base relations with the original base relations T_1 and T_2, we can derive that the update is the deletion of tuple (a, b) from T_1. This is a possible translation of the view deletion. In general, each of the solutions to the CSP corresponds to a possible translation of the view deletion.

Semantic constraints can also be used to choose an appropriate solution. As an example, suppose that a deletion against the view should not be translated into deletions against relation T_1. This can be formulated in terms of two constraints: $x \neq \eta$ and $y \neq \eta$. Adding those constraints to the CSP (3), we obtain the following CSP:

$$(\{x, y, u, v\}, D_\eta, \{y = u, x \neq \eta, y \neq \eta\}, \{x = u, x = v, y = v\}).$$

There are two solutions to this CSP:

$$\mathbf{v}_3 = \{x/b, y/b', u/b', v/\eta\}, \mathbf{v}_4 = \{x/b', y/b, u/b, v/\eta\}.$$

$(\mathbf{v}_1' \cup \mathbf{v}_3)(T_1^V) = \{(a, b), (a', b')\}$ and $(\mathbf{v}_1' \cup \mathbf{v}_3)(T_2^V) = \{(b', c)\}$ are the updated base relations with respect to $\mathbf{v}_3$. Comparing the updated base relations with the original base relations T_1 and T_2, we can derive that the updates to the base relations are the modification of tuple (a', b) in T_1 to (a', b'), the deletion of tuple (b', c) from T_2 and the modification of tuple (b, c) to (b', c) in T_2.

Similarly, $(\mathbf{v}_1' \cup \mathbf{v}_4)(T_1^V)\{(a, b'), (a', b)\}$ and $(\mathbf{v}_1' \cup \mathbf{v}_4)(T_2^V) = \{(b, c)\}$ are the updated base relations with respect to $\mathbf{v}_4$. Comparing the updated base relations with the original base relations T_1 and T_2, we can derive that the updates to the base relations are the modification of tuple (a, b) to (a, b') in T_1 and the deletion of tuple (b', c') from T_2.

Suppose that the criterion of minimal updates is adopted. The solutions with minimal changes to the valuations of the variables with respect to $\mathbf{v}_1$ should be preferred. The notion of *minimal changes* may have an application-dependent meaning. Let us simply assume that minimal changes mean a minimal number of updates. Then in the above example, $\mathbf{v}_4$ is the preferred solution because the corresponding translation of the deletion request contains two updates of the base relations, whereas the corresponding translation to $\mathbf{v}_3$ contains three updates.

4.2 Translating insertion request

Consider the insertion of tuple (a', c') into V. There are two alternative ways to translate an insertion view request, dependent of the user-specified semantics. One is to translate it into modifications. The other is to translate it into insertions.

We start with the translation of the insertion request into modifications. What we need are the same as the DM-versions of the relations T_1 and T_2, except that η does not need to be included in the domains of the variables, because deletions are not considered. We call the tables the *M-versions* of the relations; M stands for modification. The domains of the variables are assumed to be specified by D. For simplicity, let us keep the valuations of t, s, z and r as defined by $\mathbf{v}_1$ and consider only conditions involving variables x, y, u and v. Since the evaluations of tuple $(t, z), (s, z)$ and (s, r) with respect to $\mathbf{v}_1$ result in tuple $(a, c), (a', c)$ and (a', c'), the associated conditions $x = u, y = u$ and $y = v$ should be satisfied; the fourth condition $x = v$ should be falsified. We can construct the following binary CSP:

$$(\{x, y, u, v\}, D, \{x = u, y = u, y = v\}, \{x = v\}).$$

Apparently, this CSP is not solvable because the given negative constraint $x = v$ can be derived from the positive constraints. Thus we cannot keep the valuations of t, s, z and r as defined by $\mathbf{v}_1$. It can easily be checked that all the CSPs constructed based on the different valuations of t, s, z and r are not solvable. That means, we cannot translate the insertion request into modifications.

Now we consider the translation of the insertion request into insertions. Suppose that we add free tuple (a', f) to T_1 and tuple (w, c') to T_2, where f and w are new variables. Since the new variables represent values of attribute B, we define the domains of the variables to be the same as the domain of attribute B, i.e., $D(f) = D(w) = Dom(B)$. The resulting relations with free tuples are called *I-versions* of T_1 and T_2; I stands here for insertion. The following table shows the C-table representing the view on the I-versions of T_1 and T_2.

$\pi_{A,C}((T_1 \cup \{(a', f)\}) \bowtie (T_2 \cup \{(w, c')\}))$

A	C	con
a	c	
a'	c	
a	c'	$b = w$
a'	c	$f = b$
a'	c'	$(b = w) \vee (f = b') \vee (f = w)$

To achieve the effect of the insertion request, we only need to make sure that the condition associated with tuple (a', c') in the C-table is satisfied, and neither of the conditions $b = w$ and $f = b$ is satisfied; otherwise (a, c') or (a', c) would be inserted as side affects. Thus we can construct the following CSP:

$$(\{f, w\}, D, \{(b = w) \vee (b' = f) \vee (w = f)\}, \{b = w, f = b\}).$$

This CSP can be transformed into three binary CSPs. One of them is obviously not solvable, because the negative constraint $b = w$ is at the same time a positive constraint. The solution to the other two binary CSPs is $\mathbf{v}_2 = \{f/b', w/b'\}$. The translation corresponding to $\mathbf{v}_2$ is the insertion of (a', b') into T_1.

5 The Transformation

The general procedure of our approach to solving the view update problem can be summarized as follows.

Algorithm 1 *Given a database I (with no dependency constraint), a view definition q and a primitive view update request μ,*

1. *Create naive tables that represent the search space of the possible translations of the view update request.*
   - *If the update request is a deletion, then let I_c be a DM-version of I with respect to q.*
   - *If the update request is an insertion, and it is intended to be translated into insertions against the base relations, then let I_c be an I-version of I with respect to q; if the insertion request is intended to be translated into modifications against the base relations, then let I_c be an M-version of I with respect to q.*

   Let the valuation of the variables introduced in I_c w.r.t. I be $\mathbf{v}$. That is, $\mathbf{v}(I_c) = I$. We call the tuple (I, q, μ, I_c) a view update specification.
2. *Compute the view on I_c and represent it using a C-table T_c, i.e., $T_c = q(I_c)$. Based on the closure property (2) of C-tables, we can infer $\mathbf{v}(q(I_c)) = q(I)$, which means that the state of $q(I_c)$ with respect to $\mathbf{v}$ is exactly the view state $q(I)$.*
3. *Let $V' = \mu(q(I))$. Construct CSPs based on V' so that for any valuation $\mathbf{v}$ of the variables in T_c, $V' = \mathbf{v}'(T_c)$ if and only if $\mathbf{v}$ is a solution to one of the constructed CSPs.*
4. *Solve all the constructed CSPs.*

We can prove that the translation procedure presented above generates all translations of a given view update request.

Theorem 1 *Let (I, q, μ, I_c) be a view update specification. The translation procedure described in Algorithm 1 generates all database instances I' satisfying $q(I') = \mu(q(I))$.*

6 Constructing CSPs

The construction of CSPs based on a given state of a C-table is a key technical problem for our approach. Given a C-table T_c, we show in the following how to construct one or a few CSPs based on a state V of the C-table so that for any valuation $\mathbf{v}$ of the variables in T_c, $V = \mathbf{v}(T_c)$ if and only if $\mathbf{v}$ is a solution to one of the constructed CSPs. A valuation $\mathbf{v}$ is said to be *partial* with respect to $\mathbf{V}$ if for some variable $x \in \mathbf{v}$, $\mathbf{v}(x) = \eta$.

Algorithm 2 *Let V be a state of a C-table $T_c = (T, \phi)$ where table T is defined on a set X of attributes and $V = \mathbf{v}(T_c)$ for some valuation $\mathbf{v}$. A constraint satisfaction problem $(\Gamma, D, \mathcal{C})$ based on view state V can be constructed in the following way:*

- *Γ is the set of variables in T_c,*
- *D is the function defining the domain of the variables in T_c,*
- *If V is non-empty, and there is some partial valuation $\mathbf{v}_p$ that assigns values only to the variables in T such that $V \subseteq \mathbf{v}_p(T) \in rep(T)$, then $\mathcal{C}$ is the smallest set satisfying the following conditions:*

1. *Let us define a subset of table T as follows:*

$$S(T, V, \mathbf{v}_p) = \{t \in T \mid \exists t' \in V \text{ so that } t' = \mathbf{v}_p(t)\}. \quad (4)$$

For each tuple $t \in V$, let $t_1, ..., t_k \in S(T, V, \mathbf{v}_p)$ be all the tuples such that $\mathbf{v}_p(t_1) = ... = \mathbf{v}_p(t_k) = t$. Then the following constraint is in $\mathcal{C}$:

$$\vee_{i=1}^{k}(\phi(t_i) \wedge (\wedge_{A \in X} t_i(A) = t(A))).$$

2. *For each $t \in T - S(T, V, \mathbf{v}_p)$, $\neg\phi(t)$ is in $\mathcal{C}$.*

If V is empty, then $\mathcal{C}$ is the smallest set so that for each $t \in T$, $\neg\phi(t) \in \mathcal{C}$.

We can prove that the CSPs constructed based on Algorithm 2 have the required property.

Theorem 2 *Let V be a state of a C-table $T_c = (T, \phi)$, where T is defined on a set X of attributes. For any valuation $\mathbf{v}$ of the variables in T_c, $\mathbf{v}(T_c) = V$ if and only if $\mathbf{v}$ is a solution to one of the constructed CSPs based on Algorithm 2.*

7 Discussions

Our approach contains computing views on a C-table, constructing CSPs based on a state of a C-table and solving CSPs. The time/space complexity of computing a view on a C-table is polynomial in the size of the base relations [4]. According to Definition 6, there may be multiple CSPs constructed based on the same state V of a C-table $T_c = (T, \phi)$, due to the multiple partial valuations $\mathbf{v}_p$ satisfying the condition $V \subseteq \mathbf{v}_p(T) \in rep(T)$. Let n denote the number of variables in table T and a denote the maximal size of the domains of the variables. Then the number of the CSPs can be a^n. Thus the time complexity of constructing CSPs based on the given state of the C-table T_c is $O(a^n)$. Let the number of constraints in the constructed CSPs be e. Then the time complexity of solving the CSPs is $O(a^n e)$. Thus our approach has a bound that is exponential in the number of variables.

To handle semantic constraints, we can compute the enforcement of the constraints on the naive tables obtained in the first step of Algorithm 1. The result can be represented using a set of Horn tables. For equality generating dependencies, the enforcement has polynomial bounds, but tuple generating dependencies may cause exponential growth of the local conditions [4]. Thus in general, our approach has a bound that is exponential in the size of the base relations when semantic constraints are considered. This approach is more suitable for those applications where the number of variables introduced to the base relations can be limited and the dependencies are equality generating rather than tuple generating.

8 Summary

In this paper, we study a new approach to view update translation. The basic idea is to transform the problem into constraint satisfaction problems. We provide sound and complete transformation for updates of PSUJ views. This is, to our knowledge, the first reported attempt in this direction. The formal device we use is conditional table described in [6]. The approach is declarative, which makes it possible to modify the problem formalization without changing the underlying problem solver. For example, one can add or change semantic information about the intended updates on the database by modifying the CSPs; the underlying CSP solvers remain the same. It is possible to make use of the rich results of the CSP research to design algorithms for solving the view update translation problems. The approach is dependent on the current instances of the database and the view. Thus it is better applied in view update time, rather than in view definition time. The proposed approach should be adopted as a complement to the previous approaches in those cases where the view updates do not satisfy the sufficient conditions for exact and unique translations [2], and where suggestions for possible translations of the view updates are highly desirable.

Acknowledgements

This work has been supported in part by Swedish Research Council for Engineering Sciences (TFR). We would like to thank the referees for their invaluable comments on the previous version of this paper.

REFERENCES

[1] Francoise Bancilhon and N. Spyratos, 'Update semantics of relational views', *ACM Transactions on Database Systems*, **6**(4), 557–575, (1981).

[2] Umeshwar Dayal and Philip A. Bernstein, 'On the correct translation of update operations on relational views', *ACM Transactions on Database Systems*, **7**(3), 381–416, (1982).

[3] Antonio L. Furtado and Marco A. Casanova, 'Updating relational views', in *Query Processing in Database Systems*, eds., Won Kim, David S. Reiner, and Don S. Batory, 127–144, Springer-Verlag, Berlin, (1985).

[4] G. Grahne, *The Problem of Incomplete Information in Relational Databases*, Springer-Verlag, 1991.

[5] A. Gupta and I. S. Mumick, 'Maintenance of materialized views: Problems, techniques and applications', *IEEE Database Engineering Bulletin*, **18**(2), 3–18, (1995).

[6] Tomasz Imielinski and Witold Jr. Lipski, 'Incomplete information in relational databases', *Journal of the ACM*, **31**(4), 761–791, (October 1984).

[7] A. M. Keller, 'Algorithms for translating view updates to database updates for views involving selections, projections, and joins', in *4th ACM Symposium on Principles of Database Systems*, ed., ACM, pp. 154–163, New York, (1985). ACM Press.

[8] A. M. Keller, 'The role of semantics in translating view updates', *IEEE Computer*, **19**(1), 63–73, (January 1986).

[9] A.M. Keller, 'Choosing a view update translation at view definition time', in *Proceedings of the 12th International Conference on Very Large Data Bases - 1986*, eds., W. Chu, G. Gardarin, and S. Ohsuga, pp. 467–474. Morgan Kaufmann Publishers, (1986).

[10] S. Kleene, *Introduction to Metamathematics*, Van Nostrand, Princeton, 1952.

[11] R. Langerak, 'View updates in relational databases with an independent scheme', *ACM Transactions on database systems*, **15**(1), 40–66, (1990).

[12] Y. Masunaga, 'A relational database view update translation mechanism', in *Proceedings of the 10th International Conference on Very Large Data Bases - 1984*, eds., U. Dayal, G. Schlageter, and L. Seng, pp. 309–317, Palo Alto, CA, (August 1984). Morgan Kaufmann Publishers.

[13] Hua Shu, 'Using constraint satisfaction for view update translation', Technical report, DBIS, Institute of Computer Science, University of Hannover, (January 1998).

[14] Edward Tsang, *Foundations of Constraint Satisfaction*, Academic Press, UK, 1993.