1992

Enforceable Interdatabase Constraints in Combining Multiple Autonomous Databases

Aidong Zhang

Ahmed K. Elmagarmid
Purdue University, ake@cs.purdue.edu

Report Number:
92-008

ENFORCEABLE INTERDATABASE CONSTRAINTS IN COMBINING MULTIPLE AUTONOMOUS DATABASES

Aidong Zhang
Ahmed K. Elmagarmid

CSD-TR 92-008
January 1992

Enforceable interdatabase constraints in combining multiple autonomous databases *

Aidong Zhang and Ahmed K. Elmagarmid
Department of Computer Science
Purdue University
West Lafayette, IN 47907
{azhang, ake}@cs.purdue.edu

Abstract

In this paper, the enforceable interdatabase constraints arising in combining multiple autonomous databases are investigated. A concise declarative language, function-based dependencies, is provided for expressing these constraints. Update anomaly problems are addressed within this context. The tight coupling of update rewrite rules in each autonomous database makes possible automatic recovery from the violation of interdatabase constraints. A set of updates to be propagated which is minimal and sufficient to guarantee the maintenance of these constraints is presented. Finally, a transaction-oriented mechanism is introduced which uses the propagative transaction technique to implement the enforcement of interdatabase constraints.

1 Introduction

This paper addresses the problem of interdatabase constraints in combining multiple autonomous databases. Interdatabase constraints serve to describe the integrity constraints associated with data items in different local databases which are linked by certain relationships. The enforcement of interdatabase constraints requires that the global (combined)

*Sponsored by the Indiana Business Modernization and Technology Corporation(BMT), a PYI Award from NSF under grant IRI-8857952, grants from the AT&T Foundation, Bellcore, Tektronix, SERC, Mobil Oil and Bell Northern Research.

database satisfy a predefined set of such constraints. Enforceable interdatabase constraints are those constraints that can be automatically enforced in a database management system.

There has been a growing tendency to combine multiple local autonomous databases into one cooperative database. Autonomy is a major design factor to be considered and preserved in this type of integrated system. Insistence on the maintenance of such local database autonomy carries with it two salient complicating features:

- **Redundancy**: the same facts are already described in the schemata of two different local databases.
- **Heterogeneity**: the same facts in two different local databases appear in different formats or do not agree (have conflicts).

The reconciliation of these local characteristics gives rise to new global integrity constraints, called interdatabase constraints, which must now be enforced in a global database. Such interdatabase constraints are also necessitated by problems arising from application semantics and cooperation among local databases. In general, an integrity constraint describes an invariant relationship which should hold in a database environment. With the existence of interdatabase constraints in a global database, locally consistent transactions may generate global inconsistencies. For example, if there exists an interdatabase constraint between a data item d and other data items in different local databases, an update on d locally may produce an inconsistency (an "update anomaly") in the global database. To avoid such global inconsistencies in transactions, the enforcement of interdatabase constraints must be meticulously observed. Lacking a comprehensive understanding of the dependencies in question, individual application programmers are not in a position to assume this responsibility. Another mechanism must therefore be provided for the enforcement of interdatabase constraints in combining multiple local autonomous databases.

The importance of interdatabase constraints in a distributed database environment has recently begun to be appreciated. Rusinkiewicz and Sheth [SR90] introduced the concept of

interdependent data to describe the interdatabase constraints found in multidatabases, federated databases, and heterogeneous distributed databases. Informally, interdependent data is defined as two or more data items stored in different databases that are related through an integrity constraint that specifies their mutual dependency and consistency requirements.

The methodology of interdatabase constraints enforcement has also been the subject of recent attention. A particularly interesting problem is integrity constraints recovery. Through this approach, the violation of an integrity constraint does not lead inevitably to rejection of the operation in question. Instead, the database management system generates additional operations such that the constraint is satisfied, thus automatically enforcing integrity constraints. Research on this technique is in its early stages. In [G91], Garcia-Molina has argued that global integrity constraints should in some manner be treated locally. Rusinkiewicz and Sheth [RS91] proposed the Polytransaction approach to managing interdependent data. A Polytransaction is defined as a "transitive closure" of a transaction submitted to an interdependent data management system. It has not, however, been demonstrated that the Polytransaction model can be generally applied to various types of interdatabase constraints which may or may not conform to the "transitive closure" definition. More precise work need to be done.

In our opinion, the automatic enforcement of integrity constraints must grow out of the intrinsic characteristics of those constraints. While integrity constraints stipulate fixed relationships among data, they do not prescribe remedial action in the face of a violation. The specific enforcement of integrity constraints must be accomplished through a system of imperative rules. From this perspective, constraints are higher-level declarative specifications, and rules are lower-level operations. A constraint may imply a set of possible actions, while a rule provides a means of selecting among these alternatives. In order to automatically enforce interdatabase constraints, the constraint specification must be supported by a set of rules that define the appropriate operational responses.

In order to make the maintenance of interdatabase constraints transparent to users, the database management system must incorporate the necessary functions into transactions.

In this paper, we identify a specific class of interdatabase constraints, expressed by function-based dependencies, which can be automatically enforced by tight coupling of update rewrite rules in a database. Tight coupling refers to the integration of a rule system into a data manager [SH88]. A transaction-oriented mechanism which is similar to Polytransaction and uses the propagative transaction technique to implement the automatic enforcement of interdatabase constraints is proposed. We will concentrate on discussing the key properties of this transaction model.

The structure of the paper is as follows. In Section 2, a basic model that grounds the later discussion is described. Section 3 introduces the definition of function-based dependencies, which serves as a concise declarative language for expressing enforceable interdatabase constraints. We discuss in Section 4 the update anomaly problems involved in interdatabase constraints. In Section 5, we present CHASE procedures for maintaining these constraints. Section 6 provides a computing algorithm of the minimal cover set of a transaction which is sufficient to maintain function-based dependencies. Finally, an extended transaction model for enforcing function-based dependencies is set forth in Section 7.

2 The model

The environment to be considered in this paper is the combination of multiple local autonomous databases. In this section, we will present a more refined model and introduce notations that will be used throughout this paper.

Our model is a collection of autonomous local databases LDB_1, LDB_2,..., LDB_m which are coupled together without a global schema. Each local database has its own data model. We will simply use MDB to refer to the collective model and $MDBS$ to refer to the management system of MDB. The databases LDB_1, LDB_2,..., LDB_m may be located either in the same or in disparate local sites; in the latter case, the local databases are distributed. In either case, the theory of interdatabase constraints (to be introduced in the next section) remains applicable.

Each LDB_i consists of a set of data items which is denoted as D_i[1]. MDB contains the set of all data items D, with $D = \bigcup_{i=1}^{m} D_i$. Without loss of generality, we assume that the sets of data items in different local databases are syntactically disjoint. That is, $D_i \cap D_j = \emptyset$, for $i \neq j$. In other words, we require naming uniqueness in MDB. This may be easily accomplished through the use of site numbers to differentiate data items in separate local databases. With each data item d in D, we associate a set of values, called the domain of d and written $DOM(d)$, which defines the value domain of d.

The state of a local database LDB_i (indicated by LS) refers to the values of D_i in LDB_i. Similarly, the state of MDB (indicated by MS) is the combination of all local database states. In MDB, consistency must be maintained on both the local and global levels. A local database D_i is locally consistent whenever its state satisfies all of its local integrity constraints. Global consistency requires that the state of MDB satisfies global integrity constraints.

3 Declarative expression of interdatabase constraints

Generally speaking, interdatabase constraints are the statements that describe the integrity constraints on data items among local databases in a MDB environment. These constraints highlight the variety of interdependent relationships among data items. In this section, we propose a declarative language, which we term function-based dependency (**fbd**), to express enforceable interdatabase constraints.

We associate with each data item a domain of abstract symbols (called an abstract domain (ADOM)), in addition to its value domain. Without loss of generality, we assume the abstract domains of data items to be disjoint. A function v is defined to map each data item to a value in its abstract domain. We now give a formal definition of function-based dependency, as follows:

[1]A data item may be a relation, an attribute of a relation, or other component.

Definition 1 *(**Function-based dependency**)*

A function-based dependency **fbd** *is a bipartite template statement consisting of an hypothesis and a conclusion, and is written*

$$fbd : (u_1, u_2, ..., u_n) \mapsto_F u_{n+1}, \tag{1}$$

where fbd is the name of the **fbd**, $u_i = v(d_i), i = 1, ..., n$, *and* $u_{n+1} = v(d_{n+1}) = F(u_1, ..., u_n)$, *with* $d_1, d_2, ..., d_n, d_{n+1}$ *being different data items which are not in the same local database of* MDB *and* F *being a n-ary function mapping* u_1,...,u_n *to* u_{n+1}.

The semantics of function-based dependency are straightforward. Let **(1)** be an **fbd** defined on MDB, and $v(d_1)$,$v(d_2)$,...,$v(d_n)$,$v(d_{n+1})$ satisfy:

$$v(d_{n+1}) = F(v(d_1), ..., v(d_n)). \tag{2}$$

MDB satisfies the function-based dependency **(1)** if, whenever the different data items d_1,d_2,...,d_n and their values can be found in MDB, there exists a data item d_{n+1}, with its value in MDB, and the values of $d_1, ..., d_n, d_{n+1}$ satisfy **(2)**.

The definition of $(v(d_1), ..., v(d_n)) \mapsto_F v(d_{n+1})$ implies one of the following mapping relationships:

- **Partial mapping:** If there are values $w_1, ..., w_n$ of $d_1, ..., d_n$ in MDB, then there exists a value w_{n+1} of d_{n+1} such that $w_{n+1} = F(w_1, ..., w_n)$. The mapping may not be onto; it is not necessary for every value w_{n+1} of d_{n+1} to have corresponding values $w_1, ..., w_n$ of $d_1, ..., d_n$ with $w_{n+1} = F(w_1, ..., w_n)$.

- **Surjective mapping:** The values of d_{n+1} are uniquely defined by the values of $d_1, ..., d_n$; namely, the mapping is onto. Surjective mapping defines an extended derivation relationship.

- **Bijective mapping:** In case of $n = 1$, the existence of value w_1 of data item d_1 determines the existence of value w_2 of data item d_2, and vice versa. A function-based dependency $(v(d_1)) \mapsto_F v(d_2)$ that is bijective implies that there must be another $(v(d_2)) \mapsto_{F_1} v(d_1)$ defined on MDB, forming a "dual pair."

Function-based dependencies raise two areas of concern: syntactic circulation and semantic circulation. Syntactic circulation refers to the circular definition of function-based dependencies, such as replicated data. Semantic circulation refers to the semantic inconsistency of function-based dependencies. A set Σ of **fbds** is semantically inconsistent if there are conflicting functions defining Σ. A semantically inconsistent set of **fbds** cannot be satisfied by any state of MDB (Section 6 presents a detailed discussion of this point).

Function-based dependencies, which express enforceable interdatabase constraints in $MDBS$, provide a useful and powerful descriptive tool. They are effective in describing with exactitude those interdatabase constraints which can be automatically enforced in $MDBS$. Most relationships among data items can be described functionally, while some nonfunctional relationships can be symbolically transformed into functional relationships. We will use function-based dependencies to refer to the enforceable interdatabase constraints in $MDBS$ in the rest of this paper.

We will now illustrate the usefulness of function-based dependencies through some application examples.

Application 1 *(Derived data)*

Derived data in a MDB environment refers to data derived from source databases and stored in other databases. The dependency relationship of derived data is static and unique; the source databases are the determinant elements and the databases which store the derived data are dependent elements (surjective mapping). A derivation relationship does not necessarily imply interdependency unless there exists a consistency requirement between the source data and the derived data such that the derived data must be updated at some point after an update of the source data. Materialized view is a typical example of derived data.

Let $d_1, d_2, ..., d_l$ be data items in MDB. The constraint relationship indicating that the values of a data item d_{l+1} are derived from the values of data items d_1,d_2,...,d_l can be expressed as:

$$(u_1, u_2, ..., u_l) \mapsto_F u_{l+1},$$

where $u_1, u_2, ..., u_l, u_{l+1}$ are the abstract symbols of $d_1, d_2, ..., d_l, d_{l+1}$ and $u_{l+1} = F(u_1, u_2, ..., u_l)$, with $F(u_1, u_2, ..., u_l)$ being a function that defines the semantics between the values of d_1, d_2,...,d_l and d_{l+1}.

Application 2 *(Replicated data)*

Replicated data in a MDB environment refers to identical copies of data items stored in two or more local databases. The constraint relationship of replicated data can be expressed as follows:

The data items d_1 and d_2 are replicated in MDB if and only if $(u_1) \mapsto_F u_2$ and $(u_2) \mapsto_F u_1$, where u_1,u_2 are abstract symbols of d_1 and d_2 and F is an identity function (after a suitable extension of renaming).

Application 3 *(Vertical fragmentation)*

Vertical fragmentation of a relation R produces the fragments R_1,R_2,..., R_r, each of which contains a subset of R's attributes as well as a primary key of R. The objective of vertical fragmentation is to partition a relation into a set of smaller relations, allowing many of the user applications to run on only one fragment. The constraint relationships among R_1,R_2,...,R_r can be expressed as:

$$(u_i) \mapsto_{F_{ij}} u_j, \ for \ 1 \leq i, j \leq r,$$

where u_1,u_2,...,u_r are the abstract values of R_1,R_2,..., R_r, and F_{ij} is the function mapping an abstract symbol of R_i to an abstract symbol of R_j.

4 Update anomalies

In an MDB environment, a user may focus only on the schemas of a few local databases or even on a partial schema of a local database. Thus, it is very difficult for users to be aware of interdatabase constraints while writing either global transactions (accessing data at two or more sites) or local transactions (accessing data at one site). Such user disregard of global

integrity constraints may result in update anomalies which violate the consistency property of transactions.

An update, in this paper, refers to any database operation that changes the database state; more specifically, insertion, deletion, or modification. We term a data item d independently updatable if the updates on d preserve local database consistency. When data updates violate interdatabase constraints, update anomalies may arise in MDB . We define an update anomaly as follows:

Definition 2 *(Update anomaly)*

A data item d has an update anomaly if and only if there exists an update up *such that:*

(a) d is independently updatable with respect to up*; and*

(b) the update up *on d violates interdatabase constraints.*

Definition 2 implies that, if d has an update anomaly, then there must be an interdatabase constraint relating d with other data items in different local databases. Update anomalies are further typified as insertion anomalies, deletion anomalies, or modification anomalies. The following example illustrates this definition.

Example 1 *Assume a data item GRADUATE-STUDENT.NAME in* LDB_i *and another data item TEACHING-ASSISTANT.NAME in* $LDB_j (i \neq j)$*. The constraint specified by* v*(TEACHING-ASSISTANT.NAME)* $\mapsto_F$ v*(GRADUATE-STUDENT.NAME), with F being a function mapping the name of each teaching assistant to the unique name of a graduate student, is enforced in MDB. Changes made in* LDB_i *only, such as the insertion of a new teaching assistant or the modification of an existing teaching assistant's name, will violate the given function-based dependency. Note that both data items are independently updatable.*

The relationship between updates and function-based dependencies merits further analysis. We will now examine the effect of updates on function-based dependencies. For convenience, we will use $H \mapsto_F c$ to denote a function-based dependency in MDB. Let us consider the direct effect on $H \mapsto_F c$ of the execution of an update *up* on a data item d in

MDB. There are three possible outcomes: (a) $ADOM(d) \cap H = \emptyset$ and $c \notin ADOM(d)$; (b) $ADOM(d) \cap H \neq \emptyset$; or (c) $c \in ADOM(d)$. Case (a) is not problematical; $H \mapsto_F c$ is not violated. In instance (b), since the update *up* on d changes the value of d, the case violates $H \mapsto_F c$. In case (c), several analyses are possible. If $H \mapsto_F c$ is a surjective mapping, then the update on d should not be allowed. This "abnormal update" is simply denied; such unexecutable updates will not be further considered here. If $H \mapsto_F c$ is a partial mapping, then the update on d is allowed only if it will not violate $H \mapsto_F c$. For example, the insertion of a new value into d would be allowed. If $H \mapsto_F c$ is a bijective mapping, the direct affect of the update *up* on d could be equivalently considered from the dual **fbd**[2] of $H \mapsto_F c$, and the effect of *up* on the dual is similar to case (b). In conclusion, we set forth the following proposition:

Proposition 1 *Assume that no abnormal updates are executed in MDB. An* **fbd** $H \mapsto_F c$ *is violated if and only if there exists an update on* d *and* $ADOM(d) \cap H \neq \emptyset$.

Proposition 1 gives a general rule for deriving invalidating operations of function-based dependencies. Recovery from the violation of function-based dependencies will be discussed in the following sections.

5 CHASE procedures

In this section, we formally define the basic steps of recovery from the violation of function-based dependencies by means of update rewrite rules. The implementation of recovery through these procedures will be discussed in the next section.

When updates occur to a database, one or more function-based dependencies may be affected. Two recourses are possible in the event of the falsification of function-based dependencies: (1) denial of the operation which yielded the violation; or (2) generation of

[2] As discussed before, assuming $H = (v(d_1))$, the dual **fbd** of $H \mapsto_F c$ is $(c) \mapsto_{F_1} v(d_1)$ with a well-defined function F_1.

additional operations such that the function-based dependencies are satisfied. The latter approach, termed integrity constraints recovery, is the focus of this paper.

We support each function-based dependency by tight coupling of update rewrite rules into each local database. These rules uniquely define the actions needed to enforce the semantics of the related function-based dependencies. Each rule is triggered by update events on data items referred by the hypothesis of an **fbd** and produces an update on the data item referred by the conclusion of the **fbd**. Theoretically, a function defining an **fbd** uniquely determines the relationship between the hypothesis and the conclusion of the **fbd**. Consequently, update rewrite rules can be uniquely determined. A practical method for the construction of update rewrite rules merits further development. In this paper, we assume that such update rewrite rules are available.

The CHASE concept was introduced by Aho, Beeri, and Ullman [ABU79] and extended to join dependencies by Maier, Mendelzon, and Sagiv [MMS79]. Originally, the CHASE process was formulated to test the implication of dependency relationships. Here, we extend CHASE concept to the enforcement of function-based dependencies. Letting $T = \{up_1, ..., up_l\}$ represent a sequential set of updates, we define three different CHASE procedures, as follows:

5.1 CHASE-checking

Let MS be a state of MDB. A procedure checking whether MS satisfies a set Σ of **fbds** is denoted by $CHASE^C_\Sigma(MS)$. If MS satisfies Σ, then $CHASE^C_\Sigma(MS) = MS$; otherwise, $CHASE^C_\Sigma(MS)$ returns an error message. It is obvious that $CHASE^C_\Sigma(MS)$ terminates in finite steps if Σ is finite.

5.2 CHASE-triggering

The set of updates generated after the execution of T with respect to Σ and update rewrite rules is denoted by $CHASE_\Sigma(T)$, and the powers of $CHASE_\Sigma$ are defined as follows:

Let $\quad CHASE_\Sigma = CHASE_\Sigma \uparrow 0,$

for n being a positive integer,

$$CHASE_\Sigma \uparrow n = CHASE_\Sigma(CHASE_\Sigma \uparrow (n-1)).$$

5.3 CHASE-executing

CHASE-executing process with respect to a single fbd:
Let $fbd : (v(d_1),...,v(d_n)) \mapsto_F v(d)$ be a function-based dependency enforced in MDB. CHASE-executing with respect to fbd after the execution of T is the process of updating d as long as fbd is not satisfied. For example:

Let w_1,...,w_n be values of d_1,...,d_n, respectively, and let there be no value w of d such that $w = F(w_1,...,w_n)$ in MDB. Update d such that a value w' can be found and $w' = F(w_1,...,w_n)$. Incidentally, this update operation is based on an update rewrite rule triggered by T.

Let MS be the state of MDB after the execution of T. The new state of MDB that results from CHASE-executing with respect to fbd and T is denoted by $CHASE^T_{fbd}(MS)$. Since the updates applied by $CHASE^T_{fbd}(MS)$ are finite and uniquely determined by update rewrite rules, the process terminates at this point.

Lemma 1 *$CHASE^T_{fbd}(MS)$ satisfies fbd.*

Proof: This lemma is straightforward from the definition of $CHASE^T_{fbd}(MS)$.

Lemma 1 implies that every function-based dependency is dynamically recoverable.

CHASE-executing procedure with respect to a set of fbds:
Assume a finite set of Σ of **fbds** in MDB. In this case, CHASE-executing with respect to Σ after the execution of T results in repeatedly CHASE-executing with respect to each **fbd** in Σ, as long as possible.

Similarly, the new state of MDB that results from CHASE-executing with respect to a set Σ of **fbds** after the execution of T is denoted by $CHASE_{\Sigma}^{T}(MS)$.

Theorem 1 *$CHASE_{\Sigma}^{T}(MS)$ satisfies all* **fbds** *in Σ.*

Theorem 1 follows directly from Lemma 1.

The CHASE procedures defined in this section allow $MDBS$ to make additional updates as indicated by update rewrite rules and thus recover from the violation of function-based dependencies. This technique is known as "update propagation." We will discuss the integration of CHASE procedures into a transaction model in Section 7.

6 Reasoning with function-based dependencies

In this section, we analyze the sufficiency and minimality of update propagation. It is necessary to determine the minimal update propagating activities required to maintain the consistency of MDB when a transaction is executed. Update rewrite rules provide descriptions of updates on data items and corresponding updates to be generated on other data items linked by dependency relationships. These direct propagations, however, may not be sufficient in some instances. The following example is illustrative:

Example 2 *Let d_1, d_2, d_3, d_4, d_5 be data items in MDB. Assume that $fbd_1 : (u_1, u_2) \mapsto_{F_1} u_3$ and $fbd_2 : (u_3, u_4) \mapsto_{F_2} u_5$ are enforced in MDB, where u_1,u_2,u_3,u_4,u_5 are abstract values of d_1,d_2,d_3,d_4,d_5 and $u_3 = F_1(u_1, u_2) = u_1 + u_2$, $u_5 = F_2(u_3, u_4) = u_3 + u_4$. A transaction updating d_1 would violate fbd_1 and therefore the value of d_3 must be updated to recover fbd_1. Furthermore, fbd_2 would be violated by the update on d_3, and the value of d_5 has therefore to be updated to recover fbd_2. That is, the violation of an* **fbd** *may, by propagation, cause the violation of other* **fbds** *when compensating actions are taken to recover the* **fbd**.

Let T be a transaction. Without loss of generality, we assume that T is a sequential set of updates on a set of data items $D_T = \{d_1, ..., d_l\}$. We here assume that there are no circulation problems with a given set Σ of **fbds**; this issue will be addressed later. An **fbd** is said to be propagately violated while T is executed if it is violated either by the execution of T or by compensating actions taken to recover the **fbds** directly violated by the execution of T. The set of **fbds** which may be propagately violated in the course of the execution of T is termed the "cover set of T" (denoted by *cover*(T)). The definition of *cover*(T) is as follows:

Definition 3 *(cover(T)):*
The cover set of T is defined as all **fbds** *that are propagately violated while T is executed.*

We define an operation $+$ on the set D_T of data items as:

$$D_T^+ = \{H \mapsto_F c : H \mapsto_F c \in \Sigma \text{ and } ADOM(d_i) \cap H \neq \emptyset \text{ for } 1 \leq i \leq l\}.$$

D_T^+ contains the set of **fbds** in which the hypothesis of each member is conjoint with the abstract domain of a member in D_T. In other words, D_T^+ defines the set of **fbds** to be directly violated by the execution of T.

Alternatively, *cover*(T) may be defined as follows:

Definition 4 *The cover(T) with respect to Σ is computed as the union of the following hierarchy:*

$$L_0 = D_T^+,$$

for $n > 0$, we have:

$$L_n = \{H \mapsto_{F_2} v(d) : H' \mapsto_{F_1} v(d') \in L_{n-1} \text{ and } H \mapsto_{F_2} v(d) \in \Sigma - \bigcup_{i=0}^{n-1} L_i \text{ and } ADOM(d') \cap H \neq \emptyset\},$$

then $cover(T) = \bigcup_{i=0}^{\infty} L_i$.

The computation of cover(T) terminates when $\exists n \geq 0$, $L_n = \emptyset$.

Definition 4 provides an algorithmic method to compute *cover*(T). The following theorem indicates that the algorithm always halts.

Theorem 2 *(Termination)*

Given Σ as a finite set of **fbds** *to be defined on MDB, the computation of $cover(T)$ by Definition 4 terminates.*

Theorem 4 is correct in that there is no **fbd** which can appear in both L_i and L_j, where $i \neq j$, in the computation of $cover(T)$.

By Proposition 1, we see that $cover(T)$ as constructed by Definition 4 contains only those **fbds** which will be propagately violated while T is executed. Lemma 2 indicates that such a construction includes all **fbds** to be propagately violated. Consequently, Definition 3 and Definition 4 are equivalent.

Operation $*$ on a set I of **fbds** is defined as constructing the data items referred by the conclusions of the **fbds** in I. That is:

$$I^* = \{d : H \mapsto_F v(d) \in I\}.$$

Lemma 2 $[cover^*(T)]^+ \subseteq cover(T)$.

Proof: Assume that $cover(T)$ is computed by Definition 4 and $H \mapsto_F c \in [cover^*(T)]^+$. There then exists a d' such that $ADOM(d') \cap H \neq \emptyset$ and $d' \in cover^*(T)$. Consequently, $\exists n \geq 0$ *and* H', $H' \mapsto_F v(d') \in L_n$. If $H \mapsto_F c \in \bigcup_{i=0}^{n} L_i$, a conclusion has been reached; otherwise, $H \mapsto_F c \in L_{n+1}$. Hence, $H \mapsto_F c \in cover(T)$. □

Consequently, we have the following theorem:

Theorem 3 *(Minimality)*

Definition 4 identifies a unique and minimal cover(T) to be propagately violated while T is executed.

We now define MDB as globally consistent with respect to Σ by a fixpoint equation of $CHASE_{\Sigma}^{G}$.

Definition 5 *Let Σ be a set of* **fbds** *defined on MDB and MS be the state of MDB. MDB is globally consistent with respect to Σ if and only if:*

$$MS = CHASE^{C}_{\Sigma}(MS).$$

Assume MDB to be consistent before the transaction T is executed and MS^0 to be the state of MDB after T is executed. If the execution of T violates any **fbds**, then by automatically enforcing the violated **fbds**, we have:

$$MS^1 = CHASE^{T}_{cover(T)}(MS^0) \text{ and } MS^0 \neq MS^1.$$

At this point, MS^1 may not be a fixpoint of $CHASE^{C}_{\Sigma}$, since the effect of $CHASE^{T}_{cover(T)}(MS^0)$ may propagate and violate new **fbds**. This hierarchical update propagation and its enforcement can be described by means of CHASE-triggering and CHASE-executing:

$$\begin{aligned}
&MS^1 = CHASE^{T}_{cover(T)}(MS^0),\\
&MS^2 = CHASE^{CHASE_{cover(T)}\uparrow 0(T)}_{cover(T)}(MS^1),\\
&\ldots.\\
&MS^{n+1} = CHASE^{CHASE_{cover(T)}\uparrow (n-1)(T)}_{cover(T)}(MS^n), \qquad (3)\\
&\ldots.
\end{aligned}$$

Our goal is to find a minimal $cover(T)$ and a positive integer $n < \infty$ such that $CHASE_{cover(T)} \uparrow (n-1)(T) = \emptyset$, $MS^{n+1} = MS^n$, and $MS^n = CHASE^{C}_{\Sigma}(MS^n)$.

Lemma 3 *Given a finite set Σ of* **fbds** *which poses no circulation problems, there exists a positive integer $n < \infty$ such that $CHASE_{cover(T)} \uparrow (n-1)(T) = \emptyset$, where $cover(T)$ is computed by Definition 4.*

Proof: self-evident.

Lemma 4 *In the hierarchical update propagation of (3), if there exists a positive integer $n < \infty$ such that $CHASE_{cover(T)} \uparrow (n-1)(T) = \emptyset$, then MS^n is a fixpoint of $CHASE^{C}_{\Sigma}$.*

Proof: In (3), if $CHASE_{cover(T)} \uparrow (n-1)(T) = \emptyset$, then $CHASE^{CHASE_{cover(T)}\uparrow (n-1)(T)}_{cover(T)}(MS^n) = MS^n$. Hence $CHASE^{C}_{cover(T)}(MS^n) = MS^n$. By Definition 3, we have $CHASE^{C}_{\Sigma}(MS^n) =$

MS^n. □

Based on Lemmas 3 and 4, we have the following theorem:

Theorem 4 *Given a consistent MDB before the execution of transaction T, MDB is consistent after the execution of T if cover(T) is propagately maintained.*

Both syntactic and semantic circulation problems can be detected by means of the CHASE-triggering procedure.

Theorem 5 *If there exists a positive integer $n < \infty$ such that the execution of $CHASE_{cover(T)} \uparrow (n-1)(T)$ causes an update rewrite rule to be triggered repeatedly, then cover(T) is syntactically or semantically circular. Furthermore, if the function-based dependencies maintained by such update rewrite rules are not violated, then cover(T) is only semantically circular.*

Theorem 5 provides a dynamic verification of the consistency of a set of interdatabase constraints.

7 A transaction-oriented mechanism to enforce function-based dependencies

This section introduces a transaction-oriented mechanism to automatically enforce function-based constraints in $MDBS$. The mechanism, which is similar to the Polytransaction, incorporates updates generated by update rewrite rules into each transaction, allowing the transaction to tolerate violations of function-based dependencies. This mechanism is here termed the "propagative transaction model" (or $P^*(T)$ transaction model). We will now present the requirements that determine the correctness of the propagative transaction model.

The $P^*(T)$ transaction model supports both initial transactions and propagating transactions. Initial transactions are local or global transactions written by users, while propagating transactions are generated by $MDBS$. Whenever a transaction T is executed in MDB, the

minimal set $cover(T)$ of **fbds** must be enforced. $MDBS$ acts like a CHASE-triggering procedure, hierarchically generating additional update operations for each transaction by triggering the update rewrite rules attached to the **fbds** in $cover(T)$. Each additional update operation becomes a propagating transaction. A propagating transaction can then act as an initial transaction and generate one or more additional propagating transactions.

Let a transaction T comprise a sequential set of the update operations $\{up_1, ..., up_l\}$. We assume that a given set Σ of **fbds** is semantically consistent and that $n(< \infty)$ is a non-negative integer such that $CHASE_{cover(T)} \uparrow n(T) = \emptyset$. A descriptive definition of the $P^*(T)$ transaction model is given as follows:

Definition 6 *(Propagative transaction model)*
An $P^*(T)$ of T is a collection of transactions defined by:

$$P^*(T) = \bigcup_{i=0}^{n-1} CHASE_{cover(T)} \uparrow i(T) \cup \{T\},$$

where T is an initial transaction and the execution of T generates $CHASE_{cover(T)} \uparrow 0(T)$. Hierarchically, for $0 < i < n$, the execution of $CHASE_{cover(T)} \uparrow (i-1)(T)$ generates $CHASE_{cover(T)} \uparrow i(T)$. Each update in $\bigcup_{i=0}^{n-1} CHASE_{cover(T)} \uparrow i(T)$ becomes a propagating transaction of T.

The $ACID$ properties (Atomicity, Consistency, Isolation, Duration) of traditional transactions have been considered to be the key requirements that determine whether a transaction is a unit of consistent and reliable computation. While they are still suitable to both initial transaction and its propagating transactions, the $ACID$ properties can be relaxed for the $P^*(T)$ transaction. An $P^*(T)$ transaction must reflect the following key properties:

Property 1 *(Execution dependency)*
Every propagating transaction is initiated by an initial transaction or a propagating transaction. The execution of a propagating transaction forms a hierarchical order.

Property 2 *(Consistency with respect to* **fbds***)*
The consistency of an $P^*(T)$ transaction is concerned with satisfying three conditions:

(a) The initial transaction is a locally consistent transaction;

(b) The propagating transactions are locally consistent transactions; and

(c) The $P^*(T)$ transaction is a program preserving **fbds**.

The traditional atomicity property can be relaxed for $P^*(T)$ transactions. Both the initial transaction and its propagating transactions can commit without waiting for the others to commit. Hence, a global commitment protocol is not needed. However, to ensure that each transaction is grounded upon a consistent database, it must be prevented from viewing temporary inconsistencies arising in the propagation of an $P^*(T)$ transaction. The following property is therefore required:

Property 3 *(Partial isolation)*
An $P^*(T)$ transaction is semantic atomic[3] and, furthermore, the initial transaction or its propagating transactions occurring on adjacent levels in the execution hierarchy must be isolatable.

Property 4 *(Committing and aborting dependencies)*
The committing or aborting of an initial transaction determines the committing or aborting of its propagating transactions. The execution of an initial transaction T and its propagating transactions $S_1, ..., S_n$ satisfies: $commit(T) \Rightarrow commit(S_1) \wedge ... \wedge commit(S_n)$ and $abort(T) \Rightarrow abort(S_1) \wedge ... \wedge abort(S_n)$.

Thus, if the initial transaction commits, the $P^*(T)$ transaction also commits, and its aborted propagating transactions must be retried. On the other hand, if the initial transaction aborts, then the $P^*(T)$ transaction aborts, and there must be compensation for its committed propagating transactions. The dominant role of initial transactions is emphasized here, since they are the basic source for potential violations of function-based dependencies.

Definition 7 *A propagative transaction management scheme is correct if it guarantees propagating transactions satisfying Properties 1-4.*

[3]Semantic atomicity refers to a transaction which is neither atomic nor can it execute partially.

8 Discussion

We have here proposed a declarative language for expressing enforceable interdatabase constraints. We have also presented a theory of update propagation to achieve the automatic enforcement of function-based dependencies, along with a transaction-oriented mechanism to implement the theory. We consider the automatic enforcement of global integrity constraints a necessity for the combination of multiple autonomous databases. Our investigation represents a first step toward such a framework, which may be amplified by additional refinement. Currently, we are making a first approach to the construction of an propagative transaction management scheme in the context of our INTERBASE project.

Related issues offer fruitful avenues for future research. It would be of particular interest to extend our declarative language to include additional integrity constraints, such as relation-based constraints. The nondeterministic association between update rewrite rules and relation-based constraints must be addressed; a difficulty is posed by the fact that the enforcement of relation-based constraints may not be uniquely determined. User interference may be necessary to guide the actions along a desired course.

References

[ABU79] A. Aho, C. Berri and J. Ullman. The theory of joins in relational databases. *ACM Transactions on Database Systems*, 4(3): pp.297–314, September 1979.

[DHL91] U. Dayal, M. Hsu, and R. Ladin. A generalized transaction model for long-running activities and active databases. *Data Engineering Bulletin*, 14(1), pp.4-8, March 1991.

[G91] H. Garcia-Molina. Global consistency constraints considered harmful for heterogeneous database systems. In *First International Workshop on Interoperability in Multidatabase Systems*, pp. , 1991.

[GS87] H. Garcia-Molina and K. Salem. Sagas. In *Proc. ACM SIGMOD conf.*, May 1987.

[Had88] V. Hadzilacos. A theory of reliability in database systems. *Journal of the Association for Computing Machinery*, 35(1): pp.121–145, January 1988.

[M88] M. Morgenstern, and et al. Constraint-based systems: Knowledge about data. *Expert database systems*, pp.23-44, 1988.

[MMS79] D. Maier, A. Mendelzon and Y. Sagiv. Testing implications of data dependencies. *ACM Transactions on Database Systems*, 4(4): pp.455-469, December 1979.

[NY78] J. Nicolas and K. Yazdanian. Integrity checking in deductive data bases. *Logic and data bases*, pp.325-346, 1978.

[RS91] M. Rusinkiewicz and A. Sheth. Polytransactions for managing interdependent data. *Data Engineering Bulletin*, 14(1), pp.44-48, March 1991.

[RSK91] M. Rusinkiewicz, A. Sheth, and G. Karabatis. Specifying interdatabase dependencies in a multidatabase environment. Technical Report TM-STS-018609/1, Bellcore, March 1991.

[SF87] F. Sadri. Multi-relation dependencies. In *Information System*, 12(2), pp.145-149, 1987.

[SH88] M. Stonebreak and M. Hearst. Future trends in expert database systems. *Expert database systems*, pp.3-20, 1988.

[ML86] L. Mark, N. Roussopouloss and B. Chu. Update dependencies. In *Database Semantics(T.B. Steel, ed.)*, IFIP, pp. 303-319, 1986.

[SR90] A. Sheth and M. Rusinkiewicz. Management of interdependent data: Specifying dependency and consistency requirements. In *Proceedings of the Workshop on the Management of Replicated Data*, pp. 133–136, Houston, TX, November 1990.

[VM88] M. Vardi. Fundamentals of dependency theory. In *Trends in theoretical computer science(E. Borger, ed.)*, pp. 171-224, 1988.