A first step towards implementing Dynamic Algebraic Dependencies

N. Bidoit and S. De Amo*

LIPN – U.R.A. 1507 du CNRS
Université Paris XIII – Institut Galilée
Av. Jean Baptiste Clement, 93430 Villetaneuse, France
bidoit@lipn.univ-paris13.fr,lri.lri.fr
deamo@brufu.bitnet

Abstract

We present a class of dynamic constraints (DADs) which are of practical interest and allows one to express restrictions such as *if some property hold now, then in the past some other property should have been true*. The paper investigates in a constructive manner the definition of transaction-based specifications equivalent to DAD-constraint-based specifications. Our study shows the limitation of Abiteboul/Vianu's transaction schemas and proposes a generalization of transaction schemas based on regular expression on transactions.

Key words: Dynamic constraint, Transaction schema, Declarative and Operational specification of database schema.

1 Introduction

Integrity constraint maintenance is a major issue in advanced database applications. Integrity constraints are usually partioned in two classes: static constraints and dynamic constraints. A static integrity constraint is a condition on the current state of the database which is called legal when the condition is satisfied. Functional dependencies and dependencies [Ull, AHV] in general are static integrity constraints. A dynamic integrity constraint is a condition on a sequence of states representing the history of the database. The constraint stating that "salaries of employees should never decrease" is an example of dynamic integrity constraint. Note that this constraint does not need an explicit representation of time. Constraints dealing with explicit time are called temporal integrity constraints [Ko] and require storage of time in one way or another in the database. Note that static constraints are special cases of dynamic constraints, and temporal integrity constraints subsume dynamic constraints. Integrity constraints are declarative specifications. Constraint languages are logic-based languages: first order logic for static constraints, fragments of temporal logic for dynamic constraints. Temporal logic has already been investigated in order to define dynamic properties of databases in [C, CN, CCF, CF, LS, Li, FS]. In this paper, we focus our attention on a class of dynamic constraints called dynamic algebraic dependencies. In our framework, time is implicit.

Different approaches to maintain database consistency have been investigated. The naive approach can be described as follows: the application programs are run; constraints are checked; if the violation of one constraint is detected then the program actions on the database are rolled

*This work is partially supported by the French Projects PRC BD3 and PRC IA.

back. The weakness of this approach is twofold: checking integrity constraints is expensive although efficient methods have been proposed for static constraints [Ni] and no support is provided to the user or application programmer.

Commercial database management systems do not provide constraint checking mechanisms except for very restricted classes of constraints like keys or referential integrity. Thus, integrity enforcement management remains the task of the application programmer.

The second approach towards maintaining database consistency is based on the observation that, in many cases, constraint violation can be repaired. Intuitively, repairing consists in modifying the effect of the initial execution of the program in order to reach a legal state. Ensuring that the repaired execution is close enough to the initial execution stands among the criteria used to design repairing techniques. It is also one of the most difficult to formalize and fulfill. The technology of active databases [WC] is probably one of the major contributors to the recent tremendous interest in repairing techniques [FP].

The third approach is oriented towards providing supports to the application programmer in order to design correct application programs, that is programs whose executions always lead to legal database states. Following this approach, updates are restricted to a set of prespecified valid updates called transactions. The programmer is then asked to write applications using solely the predefined transactions. The advantage of a transaction-based approach is that it completely avoids rollback. Recently, transaction schemas [AV1, AV2, AV4, BR] have been investigated and their interaction with static dependencies has been studied [AV3, AV5]. This pioneer work focuses on static databases and static constraints only. The contribution of the present paper belongs to the transaction oriented approach to enforce dynamic constraints. It is based on and generalizes [AV1, AV2, AV3, AV4, AV5].

In this paper, we investigate the relationship between two paradigms that may be used to specify dynamic databases. The first paradigm provides *operational* specifications and is a crucial component of object oriented databases: a predefined set of admissible updates called transactions specifies the allowed state sequences (database histories) which are those and only those computed or generated by the transactions. The second paradigm is *declarative* and specifies the legal database histories by temporal logic statements.

The problem addressed here is to establish a connection between constraint-based specifications and transaction-based specifications. More precisely, given dynamic constraints expressed in a declarative language, we investigate in a constructive manner the existence of transactions such that the set of legal histories characterized by the constraints equals the set of histories generated by the transactions. When such a set of transactions exists, we say that it has the same effect as (or is equivalent to) the set of constraints. Note that the inclusion of the generated histories in the set of legal histories is sufficient to guarantee that the transactions enforce the constraints. However this inclusion is not sufficient to ensure that the transactions are able to perform any valid database update (see Figure 1 for an illustration). We insist here on the fact that the first inclusion is the most commonly studied in the literature to the detriment of the second inclusion. Our claim is that the second inclusion is very much important for application development.

Following a rollback oriented approach, checking dynamic constraints requires storing the entire database history. As a matter of fact, storing the entire database history can be avoided for Past Temporal Logic constraints [C] leading to the so-called *history-less* approach. Following this approach the database schema is augmented with a fixed number of auxiliary relations used to store informtion about the evolution of the database and needed to check the constraints. Doing this entails that checking dynamic constraints can be done just by looking at the current (last) state of the database.

The transaction oriented approach chosen here is history-less. A transaction is an update program whose input is the current state of the database, not the whole history of the database.

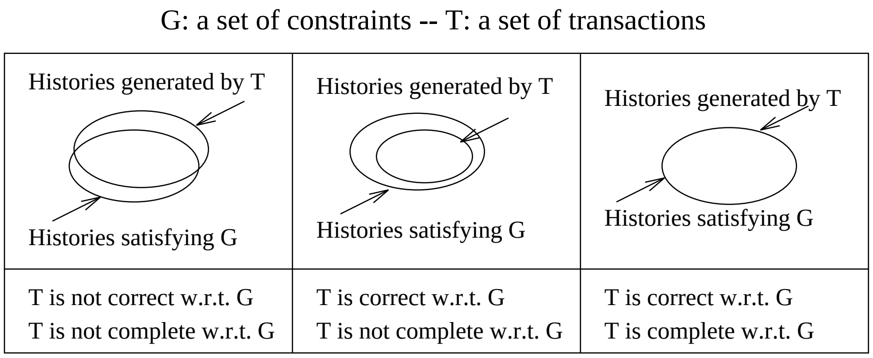

Figure 1: Correctness and Completeness

Of course like in [C], we need to keep some information about the history of the database in the current state.

Section 2 briefly reviews database terminology and transaction schemas. We also provide there notions of temporal logic which is used to specify dynamic constraints declaratively. The dynamic constraints studied in this paper are called dynamic algebraic dependencies (DADs) and are presented in section 3. They are of practical interest and allow one to express restrictions such as *if a property holds now, then in the past some other property should have hold.* An example of DAD is: "in order to be hired as a professor one should have been a student". In this section, we deal with the classical problems of consistency and implication for this class of constraints.

In Section 4 we examine two operational specifications of a dynamic database. The first one uses transaction schemas [AV2, AV3] in a very direct manner while the second one requires a generalization of transaction schemas.

A transaction schema T generates sequence of states and the transition between two states is specified by execution of one call to a transaction in T. This section starts with presenting two negative results showing the limitation of transaction schemas. Roughly speaking, it is impossible to find a transaction schema having the same effect as a set of DADs and moreover one should not expect to find a transaction schema having an effect as close as possible to the effect of a DAD schema.

Generalized transaction schemas are built up from three components: a set of relation schemas, a set of transactions, and a regular expression on transactions. The motivation is the following. A transition specified by a unique transaction call cannot (correctly) update more than one tuple at each transition. The idea for recovering arbitrary multiple changes at each transition is to allow a sequence of elementary transactions to specify a transition. However, a sequence of elementary transactions may well not preserve consistency of the database even if each transaction of the sequence does. The role of the regular expression added to the transaction schema is to restrict the transaction sequences defining transitions. Thus a generalized transaction is not any sequence of transactions but a sequence corresponding to some word of a regular language.

The main goal of sections 5 and 6 is to construct a generalized transaction schema equivalent to a given DAD schema. In section 5, we focus on elementary histories whose transitions are limited to updating at most one tuple. We show how to build a transaction schema, called elementary, generating exactly the elementary histories satisfying a fixed set of DADs. This result is obtained by adding a fixed number of historical relations to the database schema. Although this result is unsatisfactory, it provides a starting point for a general solution: an operational specification generating all legal histories should obviously be able to do it for the elementary ones.

The main result of Section 6 is a constructive specification of generalized transaction schemas equivalent to sets of DADs. This result is stated for a significant subclass of DADs.

A discussion on related work is presented before the conclusion where further research directions are outlined.

2 Preliminaries

In the following, we assume that the reader is familiar with basic database concepts [Ull, Ma, Ka] and transaction schemas [AV2, AV4]. In order to make the discussion clear, we begin by introducing well-known concepts and notations.

2.1 Relational Database and Algebra

In the sequel, we assume given four enumerable disjoint sets: a set **Attr** of attributes, a set **Dom** of constants (we assume without loss of generality that all attributes have the same domain), a set **Rel** of relation names and a set **Var** of (domain) variables. Elements of **Dom** are denoted $a, b, c, \ldots$ and elements of **Var** are denoted $x, y, z, \ldots$

It is assumed that a set Attr(R) of attributes is associated to each relation name R in **Rel**. A relation schema is given by its name R and a set of attributes Attr(R) and is denoted R(Attr(R)).

A *database schema* is a finite set of relation schemas. The notions of a *tuple* and of an *instance* over a relation schema R (and over a database schema **R**) are defined in the usual way. A *free-tuple* is simply a tuple whose values on attributes may be either constants or variables. If I is an instance over **R**, we denote by const(I) the set of constants appearing in I. The set of all possible instances over the relation schema R (resp. over the database schema **R**) is denoted by Inst(R) (resp. by Inst(**R**)).

A relational algebra expression E is defined as in [Ka] using the following operators: selection ($\sigma_{\mathrm{A=B}}$, $\sigma_{\mathrm{A=a}}$), projection (Π_{X}), natural join ($\bowtie$), renaming ($\varrho_{\mathrm{A}} \mid \mathrm{B}$), union ($\cup$) and difference ($-$). The constant query expression $\{u\}$ where u is a tuple admits {u} for answer when evaluated on any database instance. Θ denotes any relational expression which is equivalent to the *empty query*, that is, $\Theta(\mathrm{I})(\mathrm{R}) = \phi$ for all $\mathrm{I} \in \mathrm{Inst}(\mathbf{R})$ and for each $\mathrm{R} \in \mathbf{R}$. The set of attributes of the target schema of E is denoted by Tar(E) and the set of relational schemas occurring in E is denoted by sch(E). We note that $\mathrm{E} \bowtie \mathrm{F}$ is the cartesian product of E and F when $\mathrm{Tar(E)} \cap \mathrm{Tar(F)} = \phi$.

2.2 Transaction Languages

In section 4, in order to define transaction schemas we use the language SdetTL (*safe strong deterministic Transaction Language*) defined in [AV2, AV4]. For our purpose, we use a version of SdetTL (equivalent to SdetTL) which uses *parameterized relational expressions* as explicit conditions in the **while** construct. The constructs of this language are introduced in a very informal manner.

SdetTL uses some basic expressions in order to update the database: $\mathrm{ins_R}(u)$ inserts the tuple u in R, $\mathrm{del_R}(u)$ deletes the tuple u from R, $\mathrm{erase_R}$ erases the contents of relation R. In the two first expressions u may be a free tuple and then they are parameterized updates. At execution time, the variables in u are instantiated. Obviously, the language includes *composition* (denoted **;**) and *iteration* (denoted **while**). To define the syntax of the **while** construct we need to introduce *parameterized relational expressions* and *conditions*:

A *parameterized relational expression* on $\mathbf{R}$ is a relational expression where variables may occur at the place of constants. For instance, the expression $\{u\}$ where u is a free tuple over X $\subseteq$ **Attr** is a parameterized (constant query) expression whose target schema is X. Instantiating the parameter u by a tuple of constant v leads to the constant query $\{v\}$. the expression The expression $\sigma_{\mathrm{A=x}}[\mathrm{P}]$ where P$\in\mathbf{R}$ and A $\in$ Attr(P) is a parameterized expression whose parameter is x and target schema is Attr(P). Instantiating the parameter x by a constant a leads to the usual selection $\sigma_{\mathrm{A=a}}[\mathrm{P}]$.

An *atomic condition* C over $\mathbf{R}$ is an expression of the form E $\subseteq$ F where E and F are parameterized relational expressions such that Tar(E)=Tar(F). We denote by $u \in$ E (resp. $u \notin$ E) the atomic condition $\{u\} \subseteq$ E (resp. E $\subseteq$ (E $-$ $\{u\}$)). A *condition* C over $\mathbf{R}$ is a conjunct of atomic conditions. The definition of "C is satisfied by an instance I over $\mathbf{R}$" is the usual one.

A *parameterized transaction* (*p-transaction*) over $\mathbf{R}$ is a basic expression $\mathrm{ins}_{\mathrm{R}}(u)$, $\mathrm{del}_{\mathrm{R}}(u)$ or $\mathrm{erase}_{\mathrm{R}}$ or more complex expression of one the forms: $t;s$ or **while** C **do** t **done**, where t and s are p-transactions and C is a condition over $\mathbf{R}$. The free variables of $\mathrm{ins}_{\mathrm{R}}(u)$ or $\mathrm{del}_{\mathrm{R}}(u)$ are the variables occurring in u. The free variables of $t;s$ are the free variables of t or s. The free variables of **while** C **do** t **done** are the free variables of t which are not in C. The parameters $\{x_1,\ldots,x_n\}$ of a p-transaction t are made explicit by writing $t(x_1,\ldots,x_n)$. The parameters of t must include the free variables in t.

A p-transaction without parameters is called a transaction. A *call* to a p-transaction t is a transaction obtained from t by instantiating its parameters by constants in **Dom**. The set of all possible calls to t is denoted Call(t).

The semantics of SdetTL is very simple to understand. We just discuss the (deterministic) semantics of the **while** construct (see [AV2, AV4] for details). Informally, evaluating an expression of the form "**while** C **do** t **done**" is done as follows:

- For each iteration, consider the set Inst of all instantiations of the variables of the conditions C that makes C true in the current database instance. Consider the associated calls to the p-transaction t denoted Call_Inst. Then the result of executing one iteration is given by the union of the parallel executions of the transactions in Call_Inst.

- The execution of iterations stops when condition C is not satisfied by any instantiation of its variables.

Example 2.1 Let $\mathbf{R}$ be $\{$P(AB), R(AC)$\}$.

- Consider the p-transaction $t(x)$ = **while** $(x,y) \in$ R $\wedge$ $(x,y) \notin$ P **do** $\mathrm{ins}_{\mathrm{P}}(x,y)$ **done**. Execution of the call $t(a)$ inserts in P all tuples (a,y) which are in R for some $y \in$ **Dom**. Note that the execution halts after the first iteration. Assume that the instance of P before the first iteration is I(P) $= \{(a,b),(a',c)\}$ and I(R) $= \{(a,b'),(a,b''),(a'',c)\}$. After the first iteration, the instance of P is J(P) $= \{(a,b),(a',c),(a,b')\} \cup \{(a,b),(a',c),(a,b'')\} = \{(a,b),(a',c),(a,b'),(a,b'')\}$. Thus the condition in the **while** loop is false in J(P) and the iteration halts.

- Consider the transaction t = **while** $(x,y) \in$ P **do** $\mathrm{del}_{\mathrm{P}}(x,y)$ **done**. The execution of this transaction on the instance I(P) = $\{$(a,b),(b,a)$\}$ never halts because of the deterministic semantics of the **while** construct. In fact, after the first iteration, the instance of P is $(\{(a,b),(b,a)\} - \{(a,b)\}) \cup (\{(a,b),(b,a)\} - \{(b,a)\}) = \{(b,a)\} \cup \{(a,b)\} = \{(a,b),(b,a)\}$. Thus, each iteration recomputes I(P) and the condition of the **while** remains true. Thus the execution of this expression loops.

In what follows, we frequently use the **if** ... **then** ... **else** construct which can obviously be expressed in SdetTL.

2.3 Temporal Logic

As we said in the introduction, dynamic constraints are usually specified in a declarative way by temporal logic statements. We briefly give the syntax of First Order Linear Temporal Logic. We introduce only the past fragment of this language, that is, our formulae contain only *past* temporal operators. The reason for this is simple [C, CN]: the verification of a past temporal formula at a state s depends only on the database history since its creation until s.

As we soon restrict our study to a particular class of temporal formulae, we do not give the formal semantics of general temporal formulae. See [RU, Pn] for such a presentation.

The temporal language is built upon the following alphabet and symbols. Let $\mathbf{R} = \{ \mathrm{R}_1, \mathrm{R}_2, \ldots, \mathrm{R}_n \}$ be a database schema, where for each i, R_i has arity n_i. Each relation schema R_i is a predicate symbol of arity n_i. We also consider the equality predicate "=", the set **Dom** of constants, the set **Var** of variables and the usual first order symbols $\neg, \wedge, \exists$. Let us consider two temporal operators $\odot$ ("previous") and $\mathcal{S}$ ("since"). Terms are variables or constants.

The formulae are inductively defined as follows:

- If t_1 and t_2 are terms then $t_1 = t_2$ is an atomic formula.
- If $t_1, t_2, \ldots, t_n$ are terms and R is a predicate symbol of arity n then $\mathrm{R}(t_1, t_2, \ldots, t_n)$ is an atomic formula.
- If P and Q are formulae then P $\wedge$ Q and $\neg$ P are formulae.
- If P is a formula and x is a variable then $\exists x$ P is a formula.
- If P and Q are formulae then $\odot$ P and P $\mathcal{S}$ Q are formulae.
- Nothing else is a formula.

Intuitively, the temporal operators have the following semantics:

- $\odot$ P is true at some state if "in the previous state P was true".
- P $\mathcal{S}$ Q is true at some state if "sometimes in the past Q was true and since then P was true".

Other temporal operators can be derived from the previous ones:

- $\diamondminus$ F = true $\mathcal{S}$ F: " Sometimes in the past F was true".
- $\boxminus$ F = $\neg\diamondminus\neg$ F: "Always in the past F was true".

3 Dynamic Databases

Intuitively, a dynamic instance of a database stores all states of the database from its creation to the present time.

Definition 3.1 Let **R** be a database schema. A *dynamic instance* (d-instance) σ over **R** is a finite sequence of instances over **R**, $\sigma = (\sigma_0, \sigma_1, \ldots, \sigma_n)$ such that $n \geq 0$ and $\sigma_0 = \phi$. The set of all possible d-instances over **R** is denoted by d-Inst(**R**).

The fact that the first state σ_0 is empty translates the assumption that the database is empty when created. The last state of a d-instance is the current state of the database. We denote by $|\sigma|$ the size of the sequence σ. A d-instance is never a sequence of length 0, it always contains the initial empty state ϕ. We improperly say that a d-instance is *empty* when each states of its sequence is empty. The set of empty d-instances over **R** is denoted *Empty*(**R**) or simply *Empty*.

Dynamic integrity constraints are introduced in order to restrict the behavior of the database. From a general point of view, a dynamic constraint over a database schema **R** can be expressed by a closed formula of temporal logic over the predicates in **R** [C, CN, CCF, CF, LS].

Definition 3.2 A *dynamic schema* is a pair (**R**,G) where **R** is a database schema and G is a finite set of dynamic constraints over **R**. The set of dynamic instances over **R** satisfying G is denoted by Sat(**R**,G).

Definition 3.3 Let **R** be a database schema. A set $D_{\mathbf{R}}$ of d-instances over **R** is *C-generic* (where C is a finite set of constants) iff for each bijection ρ: **Dom**→ **Dom** such that $\rho\,|_C = \mathrm{id}$, we have $\sigma \in\ D_{\mathbf{R}}\ \Rightarrow \rho(\sigma) \in\ D_{\mathbf{R}}$ where $\rho(\sigma) = (\rho(\sigma_0), \ldots, \rho(\sigma_n))$ if $\sigma = (\sigma_0, \ldots, \sigma_n)$.

A *dynamic database family* over **R** is a set $D_{\mathbf{R}}$ of d-instances over **R** which is recursively enumerable and C-generic for some finite set of constants C.

A dynamic database family $D_{\mathbf{R}}$ is *bounded* if there exists $m \geq 0$ such that for each d-instance $\sigma = (\sigma_0, \ldots, \sigma_n) \in D_{\mathbf{R}}$ and for each $0 \leq i < n$, $\#\ (\mathrm{const}(\sigma_{i+1})\ -\ \mathrm{const}(\sigma_i)) \leq m$.

Sat(**R**,G) is an example of a *dynamic database family.*

In the next section we will be interested in generating dynamic database families Sat(**R**,G) by transactions. The definition of the transactions will be using auxiliary relation schemas, that is relations not in **R**. For this reason, the "generated" dynamic family will not be *exactly* Sat(**R**,G), but an *extension* of Sat(**R**,G).

Definition 3.4 Let $D_{\mathbf{R}}$ and $D_{\mathbf{S}}$ be dynamic database families. We say that $D_{\mathbf{S}}$ is an *extension* of $D_{\mathbf{R}}$ if

- $\mathbf{R} \subseteq \mathbf{S}$ and
- $D_{\mathbf{R}} = \{\sigma|_{\mathbf{R}}\ \mid\ \sigma \in D_{\mathbf{S}}\}$.

In this paper, we will study a restricted class of constraints called *dynamic algebraic dependencies.* For the sake of simplicity, we define these constraints using database algebra rather than temporal logic. The translation to temporal logic can be done in a straightforward way.

3.1 Dynamic Algebraic Dependencies (DADs)

In [AB1] we have characterized a class of dynamic constraints called anteriority dependencies and we have investigated the relationship between the declarative specification and the operational specification for a subclass of these dependencies called *dynamic inclusion dependencies.*

The purpose of this paper is the study of dynamic algebraic dependencies (DADs) which are anteriority dependencies and which subsume dynamic inclusion dependencies (DIDs) in a large extent.

Definition 3.5 A *dynamic algebraic dependency* is an expression of the form EF where E and F are relational expressions over the database schema **R** such that Tar(E) = Tar(F). The expressions E and F are respectively called the *current property* and the *past property* of the DAD EF. Indeed, EF can be expressed by the temporal formula:

$$\forall \vec{x}(\mathrm{Now}(\vec{x}) \Rightarrow \diamondsuit\ \mathrm{Past}(\vec{x}))$$

where Now($\vec{x}$) and Past($\vec{x}$) are the safe first order formulas associated to the relational expressions E and F respectively, and $\vec{x}$ denote the free variables $x_1, \ldots, x_n$ of Now (which are the free variables of Past).

We say that a d-instance $\sigma = (\sigma_0, \sigma_1, \ldots, \sigma_n)$ over **R** satisfies EF ($\sigma \models$ EF) if $\forall i \in [1 \ldots n]$, $\forall u \in$ E(σ_i), $\exists j \in [1 \ldots i-1]$ such that $u \in F(\sigma_j)$.

Example 3.1 Let **U** = {Stud(Name, Address), Prof(Name,Course)} be a database schema. The dynamic constraint saying that

in order to be a professor one should have been a student in the past

is a *dynamic algebraic dependency* over **U** denoted by EF where E = Π_{Name}[Prof] and F = Π_{Name}[Stud]. The *University* DAD schema is given by (**U**,UNIV) where UNIV denotes the above DAD EF in the remaining of the paper.

The d-instance σ presented below violates the DAD UNIV :

σ_0		σ_1		σ_2	
Stud	Prof	Stud	Prof	Stud	Prof
ϕ	ϕ	(c,d)	ϕ	(c,d)	(a,b)

The constraint UNIV can be expressed by the following temporal logic formula:

$$\forall x (\exists y \text{ Prof}(x,y) \Rightarrow \diamond \exists z \text{ Stud}(x,z))$$

A *dynamic inclusion dependency* (DID) is a DAD such that E and F are projections, i.e. E = Π_X [P] and F = Π_X [Q] for P,Q $\in$ **R**. Indeed UNIV is a DID.

We now propose two more examples:

Example 3.2 Let **U'** = {Stud(Name, Address), Prof(Name,Course), Emp(Name,Address,Dept)} be part of a university database schema. Let us assume that this university is located in LA. The dynamic constraint saying that

in order for a professor to be a member of the university administration he/she should have been a student living in LA before

is expressed by the temporal formula:

$$\forall x (\exists y \text{Prof}(x,y) \wedge \exists z \exists w \text{ Emp}(x,z,w) \wedge w = \text{"Administration"} \Rightarrow \diamond \text{Stud}(x, \text{"}LA\text{"}))$$

This formula corresponds to the *dynamic algebraic dependency* over **U'** denoted by EF where E = Π_{Name}([Prof] $\bowtie$ $\sigma_{\text{Dep="Administration"}}$[Emp]) and F = Π_{Name}($\sigma_{\text{Address="LA"}}$[Stud]).

Example 3.3 Assume that a bank decides to automatically give credit facilities to all its employees. The only reason an employee may not benefit of this advantage is if his/her name is on a blacklist of the national bank. Consider the database schema
R = {Emp(Name,Address),Cred(Name,Amount),BlackList(Name)}.
The constraint is expressed by the temporal formula:

$$\forall x (\exists y \text{Emp}(x,y) \wedge \neg \exists z \text{Cred}(x,z) \Rightarrow \diamond \text{BlackList}(x))$$

This formula corresponds to the *dynamic algebraic dependency* over **R** denoted by EF where E = Π_{Name}Emp $-$ Π_{Name}Cred and F = BlackList.

A DAD *schema* is a pair (**R**,G), where **R** is a database schema and G is a finite set of DADs over **R**. We say that a d-instance σ over **R** *satisfies* G if $\sigma \models g$, for each $g \in$ G. Recall that the set of d-instances over **R** satisfying G is denoted by Sat(**R**,G).

We now proceed to a short discussion on the relationship between the algebraic dependencies introduced in [YP] and our dynamic algebraic dependencies. [YP] defines algebraic dependencies as inclusion constraints E $\subseteq$ F on algebraic expressions[1] E and F. An algebraic dependency E $\subseteq$ F, also denoted EF, is a static integrity constraint. An instance I satisfies the static algebraic dependency EF iff the answers of the query E performed on I are among the answers of the query F performed on I also. A d-instance $(\sigma_0, \ldots, \sigma_n)$ satisfies the dynamic algebraic dependency EF iff the answers of the query E performed on σ_n can be found among the answers of the query F performed on the past states $\sigma_1, \ldots, \sigma_{n-1}$. Dynamic algebraic dependencies and static algebraic dependencies are disjoint classes of constraints although they are both based on query inclusion and share exactly the same notation. Static algebraic dependencies have the ability of expressing all kind of dependencies [Ab]. It seems that it is not the case of dynamic algebraic dependencies with respect to dynamic constraints. [AV2, AV4] exhibit subclasses of static algebraic dependencies having equivalent transaction-based specification but also subclasses of static algebraic dependencies admiting no equivalent transaction-based specifications.

3.2 The consistency problem

One can easily notice that the set of empty d-instances is always included in Sat(**R**,G). Thus, we could say that a DAD schema is always consistent. However, when Sat(**R**,G) is exactly equal to *Empty* this is not appealing to our intuition because the constraints forbid any information to enter the database. We propose the following notion of consistency:

Definition 3.6 We say that the DAD schema (**R**,G) is *consistent* when Sat(**R**,G) $\neq$ *Empty*.

The following result concerns the undecidability of consistency:

Theorem 3.1 The consistency problem for DADs is undecidable.

Proof: Let **R** be a database schema and let E,F be relational expressions over **R** such that Tar(E) = Tar(F). We prove that:

(*****) Sat(**R**,(EF)) $\neq$ *Empty* if and only if there exists I $\in$ Inst(**R**) such that I $\neq \phi$ and E(I) = ϕ.

In fact, let us assume Sat(**R**,(EF)) $\neq$ *Empty*. Thus, there exists a d-instance $\sigma = (\sigma_0, \ldots, \sigma_n)$ such that $\sigma \neq \phi$. Let i be the smallest integer such that $\sigma_i \neq \phi$. As σ satisfies (EF) and F(σ_j) = ϕ, for each $j \in [0, \ldots, i-1]$, then we conclude that E(σ_i) = ϕ.

Conversely, let us assume I $\in$ Inst(**R**), I $\neq \phi$ and E(I) = ϕ. Thus, the d-instance (ϕ,I) is not empty and satisfies (EF). Therefore Sat(**R**,G) $\neq$ *Empty*.

From (*****), we conclude that deciding consistency of EF is equivalent to deciding the *satisfiability* of the formula

$$\neg \exists x_1 x_2 \ldots x_n \varphi(x_1, x_2, \ldots, x_n)$$

over finite structures, where φ is a safe first order formula corresponding to E. This problem is well known to be undecidable, from Trakhtenbrot's theorem [LR]. $\square$

[1] E and F are SPJR-queries, i.e. union and difference are excluded

Note that this result is strongly related to the decidability of query containment. As a matter of fact, consistency for DIDs is decidable [AB1]. This is not surprising since we consider only projection expressions for current and past properties in DIDs.

A more restrictive notion of consistency called *strong consistency* is closely related with the needs of historical relations in the operational specification implementing a set of DADs.

Definition 3.7 A d-instance $(\sigma_0, \sigma_1, \ldots, \sigma_n)$ is called *trivial* w.r.t. a DAD schema ($\mathbf{R}$,G) if $\mathrm{E}(\sigma_i) = \phi$ for each current property E appearing in G and for each $i \in [0, \ldots, n]$. We denote by Tr($\mathbf{R}$,G) the set of trivial d-instances w.r.t. to ($\mathbf{R}$,G). We notice that *Empty* $\subseteq$ Tr($\mathbf{R}$,G) $\subseteq$ Sat($\mathbf{R}$,G). We say that a DAD schema ($\mathbf{R}$,G) is *strongly consistent* if there exists a non trivial d-instance satisfying G, i.e., Sat($\mathbf{R}$,G) $\neq$ Tr($\mathbf{R}$,G).

The *University* DAD schema of example 3.1 is strongly consistent. Of course, strong consistency entails consistency but the converse is false as shown by the following example:

Example 3.4 Let $\mathbf{R}$ = {P(ABC),Q(AC),R(B)} and

$$\mathrm{G} = \{(\mathrm{E}_1\ \mathrm{F}_1), (\mathrm{E}_2\ \mathrm{F}_2), (\mathrm{E}_3\ \mathrm{F}_3)\} \text{ where:}$$

$$\begin{array}{lcl} \mathrm{E}_1 = \Pi_{\mathrm{A}}\mathrm{P} & \text{et} & \mathrm{F}_1 = \Pi_{\mathrm{A}}\mathrm{Q} \\ \mathrm{E}_2 = \Pi_{\mathrm{C}}\mathrm{Q} & \text{et} & \mathrm{F}_2 = \Pi_{\mathrm{C}}\mathrm{P} \\ \mathrm{E}_3 = \Pi_{\mathrm{B}}\mathrm{P} & \text{et} & \mathrm{F}_3 = \Pi_{\mathrm{B}}\mathrm{R} \end{array}$$

G is not strongly consistent. In fact, if there exists a d-instance σ and $i \in [0 \ldots \mid \sigma \mid]$ such that $\mathrm{E}_1(\sigma_i) \neq \phi$, then $\sigma_i(\mathrm{P}) \neq \phi$ and there exists $j \in [0 \ldots i-1]$ such that $\sigma_j(\mathrm{Q}) \neq \phi$. But in this case there will be $k \in [0 \ldots j-1]$ such that $\sigma_k(\mathrm{P}) \neq \phi$. Thus, there exists an *infinite* sequence of states in $[0 \ldots i]$ for which the relation P is not empty. Contradiction. In the other hand, G is consistent because the d-instance (ϕ, σ_1) where $\sigma_1(\mathrm{P}) = \sigma_1(\mathrm{Q}) = \phi$ and $\sigma_1(\mathrm{R}) = \{(b)\}$ is not empty and satisfies G.

3.3 The implication problem

Another classical problem which arises when we study a class of constraints is the implication problem. We say that a set Σ of DADs *entails* a DAD g ($\Sigma \models g$) if for each d-instance σ which satisfies Σ we have that $\sigma \models g$. We have:

Theorem 3.2 The implication problem for DADs is undecidable.

Proof: Let $\mathbf{R}$ be a database schema and let E, F be relational expressions over $\mathbf{R}$ such that Tar(E) = Tar(F) = X. Let P, Q be two relation schemas such that P, Q $\notin \mathbf{R}$ and Attr(P) = Attr(Q) = X. We claim that: (*) {(PE),(FQ)} $\models$ (PQ) if and only if E $\subseteq$ F.

Let $\mathbf{S} = \mathbf{R} \cup$ {P,Q}. Let us assume that E $\subseteq$ F. Let σ be a d-instance over $\mathbf{S}$ such that $\sigma \models$ (PE) and $\sigma \models$ (FQ). We will show that $\sigma \models$ (PQ). Let $0 < i \leq \mid \sigma \mid$ and $u \in \sigma_i(\mathrm{P})$. Since $\sigma \models$ (PE), we can conclude that there exists $0 \leq j < i$ such that $u \in \mathrm{E}(\sigma_j)$. Since E $\subseteq$ F, we can conclude that $u \in \mathrm{F}(\sigma_j)$. Finally, since $\sigma \models$ (FQ), we can conclude that there exists $0 \leq k < j$ such that $u \in \sigma_k(\mathrm{Q})$. Thus, $\sigma \models$ (PQ).

Conversely, let us assume that E $\not\subseteq$ F. In order to show that {(PE),(FQ)} $\not\models$ (PQ), we construct a d-instance σ over $\mathbf{S}$ such that $\sigma \models$ {(PE),(FQ)} but $\sigma \not\models$ (PQ).

Since E $\not\subseteq$ F, there exists an instance I over $\mathbf{R}$ such that E(I) $\not\subseteq$ F(I). Let $u \in$ E(I) and $u \notin$ F(I). The table below defines $\sigma = (\phi, \sigma_1, \sigma_2, \sigma_3)$. In this table, R is a relation schema in $\mathbf{R}$.

	σ_0	σ_1	σ_2	σ_3
P	ϕ	ϕ	ϕ	$\{u\}$
Q	ϕ	F(I)	ϕ	ϕ
R	ϕ	ϕ	I(R)	ϕ
E	ϕ	ϕ	E(I)	ϕ
F	ϕ	ϕ	F(I)	ϕ

We can easily see that $\sigma \models \{(\mathrm{PE}),(\mathrm{FQ})\}$. But $\sigma \not\models (\mathrm{PQ})$, since $u \in \sigma_3(\mathrm{P})$ and $u \notin \sigma_i(\mathrm{Q})$, for each $i \in [0, 1, 2]$.

From (*) above, the decidability of the implication problem for DADs would entail the decidability of the containment problem for relational expressions. Using Trakhtenbrot's theorem, we can easily conclude that this problem is undecidable [Ka]. □

In [AB1] we have presented an axiomatization for strict (and non strict) DIDs. This result entails that the implication problem for this class of constraints is decidable. We focus now on the presentation of inference rules for DIDs. Proofs can be found in [AB1]. Recall that a DID EF is defined by two projections E=Π_{X}(R) and F=Π_{X}(S), thus below EF is denoted simply RS(X).

- Inclusion:
  $\forall \mathrm{Y} \subseteq \mathrm{X} \subseteq \mathrm{Attr(R)} \cap \mathrm{Attr(S)} \quad \mathrm{RS(X)} \mapsto \mathrm{RS(Y)}$
- Transitivity:
  $\forall \mathrm{X} \subseteq Attr(R) \cap Attr(S) \cap Attr(P) \quad (RS(\mathrm{X})\ and\ SP(\mathrm{X})) \mapsto RP(\mathrm{X})$
- Cyclicity:
  $\forall \mathrm{X} \subseteq \mathrm{Attr(R)} \cap \mathrm{Attr(S)}, \forall \mathrm{Y} \subseteq \mathrm{Attr(S)}$
  $(\mathrm{RS(X)\ and\ SS(Y)}) \mapsto (\forall \mathrm{Z} \subseteq \mathrm{Attr(R)} \cup \mathrm{Attr(P)} \quad \mathrm{RP(Z)})$

4 Operational Specification

In this section, we introduce two notions of operational specification. The first one and the most natural is based on Abiteboul&Vianu's transaction schemas. We define classical concepts of soundness and completeness of transaction schemas w.r.t. a set of dynamic constraints. We exhibit a necessary condition on dynamic constraint schema for the existence of a sound and complete transaction schema. It is showed that most DADs do not satisfy this condition and thus transaction schemas are not suitable for maintaining DADs in general.

For this reason, we introduce a new notion of operational specification based on generalized transaction schemas. In section 7, we will show that for a significant subclass of DADs there exists a generalized transaction schema implementing a set of these constraints.

4.1 Transaction Schemas

A *transaction schema* is a pair (**R**,T) where **R** is a database schema and T is a finite set of p-transactions over **R** [AV2, AV4].

Definition 4.1 The set of d-instances over **R** generated by the transaction schema (**R**,T) is defined by d-gen(**R**,T) $= \{\sigma \in \text{d-Inst}(\mathbf{R}) \mid \forall\ i \in [1..|\sigma|], \exists t \in \mathrm{Call(T)}, \sigma_i = t(\sigma_{i-1})\}$ where Call(T) is the set of calls to p-transactions in T. In what follows, "a call to a p-transaction in T" is abbreviated by "a transaction in T".

As the main goal of the paper is to investigate the relationship between database families specified by DAD schemas and database families specified by transaction schemas (see figure 1 in the introduction), we introduce the notion of *transaction families*. We say that a dynamic family $D_{\mathbf{S}}$ is a *transaction family* if there exists a transaction schema ($\mathbf{S}$,T) such that $D_{\mathbf{S}}$ = d-gen($\mathbf{S}$,T).

The following proposition gives a necessary condition for a dynamic family $D_{\mathbf{S}}$ to be transaction.

Proposition 4.1 If $D_{\mathbf{S}}$ is a transaction family then $D_{\mathbf{S}}$ is bounded.

Proof: This follows from the fact that the *transition sets* associated to a transaction schema in SdetTL are bounded (see [AV2, AV4]). □

Classically, we are interested in correctness and completeness. Correctness of the transaction schema ($\mathbf{S}$,T) with respect to the DAD schema ($\mathbf{R}$,G), where $\mathbf{R} \subseteq \mathbf{S}$, expresses that the d-instances generated by T are satisfying G. Completeness ensures that each d-instance satisfying G can be generated by the transactions in T. Of course a transaction schema ($\mathbf{S}$,T) "has exactly the same effect" as a DAD schema ($\mathbf{R}$,G) when both correctness and completeness hold.

Definition 4.2 Let ($\mathbf{R}$,G) be a DAD schema and let ($\mathbf{S}$,T) be a transaction schema such that the database schema $\mathbf{S}$ contains the database schema $\mathbf{R}$. Then:

- ($\mathbf{S}$,T) is *correct* w.r.t. ($\mathbf{R}$,G) if there exists an extension $D_{\mathbf{S}}$ of Sat($\mathbf{R}$,G) such that d-gen($\mathbf{S}$,T) $\subseteq D_{\mathbf{S}}$. We say that a transaction $t \in$ T is correct w.r.t. ($\mathbf{R}$,G) if ($\mathbf{S}$,$\{t\}$) is correct w.r.t. ($\mathbf{R}$,G).
- ($\mathbf{S}$,T) is *complete* w.r.t. ($\mathbf{R}$,G) if there exists an extension $D_{\mathbf{S}}$ of Sat($\mathbf{R}$,G) such that $D_{\mathbf{S}}$ $\subseteq$ d-gen($\mathbf{S}$,T).

The notion of *preservation* presented below is more natural than correctness. Intuitively, a transaction schema ($\mathbf{S}$,T) preserves a dynamic schema ($\mathbf{R}$,G) if whenever one starts with a d-instance satisfying G and uses a transaction $t \in$ T (applied on the current -last- state of the d-instance) to expand the d-instance then the resulting d-instance still satisfies G.

Definition 4.3 A transaction schema ($\mathbf{S}$,T) *preserves* a dynamic family $D_{\mathbf{S}}$ if for each d-instance $\sigma = (\sigma_0, \sigma_1, \ldots, \sigma_n) \in D_{\mathbf{S}}$ and for each $t \in$ Call(T), if $\sigma_{n+1} = t(\sigma_n)$ then $(\sigma, \sigma_{n+1}) \in D_{\mathbf{S}}$[2]. We say that ($\mathbf{S}$,T) *preserves* a dynamic schema ($\mathbf{R}$,G) if the family Sat($\mathbf{R}$,G) has an extension $D_{\mathbf{S}}$ such that ($\mathbf{S}$,T) preserves $D_{\mathbf{S}}$.

Obviously, ($\mathbf{S}$,T) preserves the family d-gen($\mathbf{S}$,T). The relationship between correctness, preservation and completeness is given by the following proposition:

Proposition 4.2 Let ($\mathbf{R}$,G) be a dynamic schema and ($\mathbf{S}$,T) be a transaction schema, where $\mathbf{R} \subseteq \mathbf{S}$.

1. If ($\mathbf{S}$,T) preserves ($\mathbf{R}$,G) and the d-instance (σ_0) satisfies G then ($\mathbf{S}$,T) is correct w.r.t. ($\mathbf{R}$,G).
2. Correctness does not entail preservation.
3. If ($\mathbf{S}$,T) is correct and complete w.r.t. ($\mathbf{R}$,G) then ($\mathbf{S}$,T) preserves ($\mathbf{R}$,G).

[2] (σ, σ_{n+1}) stands for $(\sigma_0, \sigma_1, \ldots, \sigma_n, \sigma_{n+1})$

Proof:
(1) Let $D_{\mathbf{S}}$ be an extension of Sat($\mathbf{R}$,G) such that ($\mathbf{S}$,T) preserves $D_{\mathbf{S}}$. Let $\sigma = (\sigma_0, \ldots, \sigma_n) \in$ d-gen($\mathbf{S}$,T). From the fact that $(\sigma_0) \in D_{\mathbf{S}}$, we can show by induction on n that $\sigma \in D_{\mathbf{S}}$. Thus, d-gen($\mathbf{S}$,T) $\subseteq D_{\mathbf{S}}$.
(2) To show that, let us consider $\mathbf{R} = \mathbf{S} = \{$P(AB),Q(AC)$\}$ and G $= \{ f \}$ where f is the following dynamic constraint: $\forall x(\exists y \mathrm{P}(x,y) \rightarrow \Diamond\exists z\ \mathrm{Q}(x,z))$.
Let T $= \{t(x,y)\}$ where $t(x,y)$=while Q(z,w)$\wedge\neg$ P(x,y) do $\mathrm{ins_P}(x,y)$ done. ($\mathbf{R}$,T) is correct w.r.t. ($\mathbf{R}$,G) because d-gen($\mathbf{R}$,T) $=$ *Empty* $\subseteq$ Sat($\mathbf{R}$,G). However, T does not preserve G. In fact, let $\sigma = (\sigma_0, \sigma_1)$ where $\sigma_1(\mathrm{P})=\phi$ and $\sigma_1(\mathrm{Q})=\{(\mathrm{a,b})\}$. It is clear that $\sigma \in$ Sat($\mathbf{R}$,G). Let us consider $\sigma'=(\sigma_0, \sigma_1, \sigma_2)$ where $\sigma_2 = t(c,d)(\sigma_1)$, with $c \neq a$. Then: $\sigma_2(\mathrm{P})=\{(\mathrm{c,d})\}$ and $\sigma_2(\mathrm{Q})=\{(\mathrm{a,b})\}$. The d-instance σ' does not satisfy G.
(3) If ($\mathbf{S}$,T) is correct and complete w.r.t. ($\mathbf{R}$,G) then d-gen($\mathbf{S}$,T) is an extension of Sat($\mathbf{R}$,G). On the other hand, ($\mathbf{S}$,T) preserves d-gen($\mathbf{S}$,T), thus ($\mathbf{S}$,T) preserves ($\mathbf{R}$,G). □

The relationship between constraint-based specifications and transaction-based specifications can now be stated by:

> Given a DAD schema ($\mathbf{R}$,G) is it possible to find a transaction schema ($\mathbf{S}$,T) such that ($\mathbf{S}$,T) is correct and complete w.r.t. to ($\mathbf{R}$,G), that is, does Sat($\mathbf{R}$,G) have a transaction extension ?

As we will show below (theorem 4.4) this question has a negative answer in most cases. In fact, this is the case when DADs are *positive*. We say that a DAD EF is *positive* if E does not use the difference operator. Let us take our University example as motivation:

Example 4.1 Let us consider the University DAD schema ($\mathbf{U}$,UNIV) of example 3.1. It is clear that it is consistent and positive. Consider the transaction schema ($\mathbf{U}$,T) where T is the following set of the p-transactions:

$$\begin{aligned}
\mathrm{hire}(x,y) &= \textbf{if } (\mathbf{x},\mathbf{z}) \in \mathrm{Stud} \textbf{ then } \mathrm{ins_{Prof}}(\mathbf{x},\mathbf{y}) \\
\mathrm{register}(x,y) &= \mathrm{ins_{Stud}}(x,y) \\
\mathrm{fire}(x,y) &= \mathrm{del_{Prof}}(x,y) \\
\mathrm{cancel_registration}(x,y) &= \mathrm{del_{Stud}}(x,y)
\end{aligned}$$

It is easy to check that T is correct w.r.t. ($\mathbf{U}$,UNIV) that is T generates legal d-instances.

The transaction schema ($\mathbf{U}$,T) is unable to generate all elementary d-instances satisfying UNIV. The d-instance D described below satisfies UNIV, but it cannot be generated by ($\mathbf{U}$,T).

σ_0		σ_1		σ_2		σ_3	
Stud	Prof	Stud	Prof	Stud	Prof	Stud	Prof
ϕ	ϕ	(a,b)	ϕ	ϕ	ϕ	ϕ	(a,c)

Consider the transaction schema ($\mathbf{H}$,T') where $\mathbf{H} = \{\mathbf{U}\} \cup \{$S(Name)$\}$ and T' is the set of transactions fire(x,y) and cancel-registration(x,y) together with:

$$\begin{aligned}
\mathrm{hire}'(x,y) &= \textbf{if } \mathbf{x} \in \mathrm{S} \textbf{ then } \mathrm{ins_{Prof}}(\mathbf{x},\mathbf{y}) \\
\mathrm{register}'(x,y) &= \mathrm{ins_{Stud}}(x,y)\mathrm{ins_S}(x)
\end{aligned}$$

($\mathbf{H}$,T') is correct w.r.t. ($\mathbf{U}$,UNIV) but it is unable to generate all d-instances satisfying UNIV. The d-instance σ given below (where c $\neq d$) satisfies UNIV but it cannot be generated by ($\mathbf{H}$,T').

σ_0		σ_1		σ_2	
Stud	Prof	Stud	Prof	Stud	Prof
ϕ	ϕ	(a,b)	ϕ	ϕ	(a,c) (a,d)

We cannot obtain σ_2 from σ_1 by executing only one transaction of T'. In order to generate this d-instance and more precisely in order to get the transition from σ_1 to σ_2 we must execute the *sequence* of transactions hire'(a,c) ; hire'(a,d).

Lemma 4.3 Let $\mathcal{E}$ be a set of relational expressions over $\mathbf{R}$ which do not contain the difference operator and let I be a nonempty instance over $\mathbf{R}$ such that $\mathrm{E(I)} = \phi$ for each $\mathrm{E} \in \mathcal{E}$. Then there exists an instance J over $\mathbf{R}$ such that $\mathrm{E(J)} = \phi$ for each $\mathrm{E} \in \mathcal{E}$ and such that #const(J) > #const(I).

Proof: First recall that if E is a relational expression over $\mathbf{R}$ which does not contain the difference operator then $\mathrm{E} \equiv \mathrm{E}_1 \cup \mathrm{E}_2 \cup \ldots \cup \mathrm{E}_n$ where each E_i is a relational expression containing neither union nor difference. Thus it suffices to prove Lemma 4.3 for relational expressions without union.

Let R be a relation schema such that $\mathrm{I(R)} \neq \phi$ (there exists a relation schema satisfying this condition because I is not empty). Now let u be any tuple over R such that:

- $\mathrm{const}(u) \cap (\bigcup_{\mathrm{E}\in\mathcal{E}} \mathrm{const(E)} \cup \mathrm{const(I)}) = \phi$ (i.e. the constants appearing in u appear neither in I nor in the relational expressions E of $\mathcal{E}$).
- for each A, B $\in$ Attr(R), $u(\mathrm{A}) \neq u(\mathrm{B})$.

Let J be the instance over $\mathbf{R}$ such that

$$\mathrm{J(P)} = \begin{cases} \mathrm{I(P)} & \text{if } \mathrm{P} \neq \mathrm{R} \\ \mathrm{I(P)} \cup \{u\} & \text{otherwise} \end{cases}$$

It is clear that $\mathrm{const(J)} = \mathrm{const(I)} \cup \mathrm{const}(u)$ and #const(J) > #const(I), because const(I) $\cap$ const(u)=ϕ. Let $\mathrm{E} \in \mathcal{E}$. We will show that $\mathrm{E(J)} = \phi$. To show that, let us consider the normal form of E

$$\Pi_{\mathrm{X}}((v) \bowtie \sigma_{\mathrm{C}}(\delta_{f_1}(\mathrm{R}_1) \bowtie \ldots \bowtie \delta_{f_k}(\mathrm{R}_k)))$$

where

- v is a constant tuple, v: $\mathrm{U} \subseteq \mathbf{Attr} \longrightarrow \mathbf{Dom}$.
- $\mathrm{U} \subseteq \mathrm{X}$.
- $\mathrm{U} \cap \mathrm{Tar}(\delta_{f_j}(\mathrm{R}_j)) = \phi$, for each $j \in \{1, \ldots, k\}$.
- if $\mathrm{U}_j = \mathrm{Tar}(\delta_{f_j}(\mathrm{R}_j)))$, for each $j \in \{1, \ldots, k\}$ then $\mathrm{U}_i \cap \mathrm{U}_j$ if $i \neq j$.
- C is a positive selection condition.

Let us denote the expression $\delta_{f_1}(\mathrm{R}_1) \bowtie \ldots \bowtie \delta_{f_k}(\mathrm{R}_k)$ by E_Join. In order to show that $\mathrm{E(J)} = \phi$, it suffices to verify that σ_CE_Join(J) $= \phi$. From the fact that $\mathrm{E(I)} = \phi$, we can conclude that σ_CE_Join(I) $= \phi$. Two cases arise:

- There exists $j \in \{1, \ldots, k\}$ such that $\delta_{f_j}(\mathrm{R}_j)(\mathrm{I}) = \phi$. In this case, $\mathrm{R}_j \neq \mathrm{R}$ (by assumption) and then $\mathrm{J}(\mathrm{R}_j) = \mathrm{I}(\mathrm{R}_j) = \phi$. Hence, σ_CE_Join(J) $= \phi$.

- $\delta_{f_j}(\mathrm{R}_j)(\mathrm{I}) \neq \phi$, for each $j \in \{1, \ldots, k\}$ and for each $w \in$ E_Join(I) there exists an atomic condition C' of C such that $w \not\models$C'. We have the following possibilities for the atomic conditions in C: **(a)** they are of the form A = A' where A = f_i(B) and A' = f_j(B), for i, j such that $\mathrm{R}_i = \mathrm{R}_j$ and B $\in$ Attr(R_i). This case cannot arise because for each $w \in$ E_Join(I) there exists an atomic condition C' of C such that $w \not\models$C'. **(b)** there exist atomic conditions of the form A = a, $a \in$ **Dom** or A = B, where A,B $\in$ Attr(R_i). The way u is chosen entails that it cannot satisfy these conditions. **(c)** There exists some atomic condition of the form A = B, where A $\in$ Attr(R_i) and B $\in$ Attr(R_j), $\mathrm{R}_i \neq \mathrm{R}_j$. Again, the way u is chosen entails that it cannot satisfy these conditions.

  Hence, if σ_CE_Join(J) $\neq \phi$ then there must exist a tuple in E_Join(I) satisfying C. However this is not possible, because for each $w \in$ E_Join(I) there exists an atomic condition C' of C such that $w \not\models$C'. □

We can show now that:

Theorem 4.4 If (**R**,G) is a consistent and positive DAD schema then a transaction extension of Sat(**R**,G) does not exist.

Proof: By using proposition 4.1, it suffices to show that Sat(**R**,G) has no bounded extension. From the fact that G is assumed to be consistent, we know that Sat(**R**,G)$\neq$ *Empty*. Hence, there exists $\sigma \in$ Sat(**R**,G) such that $\sigma \neq \phi$. Let $i \in [0, \ldots, \mid \sigma \mid]$ be the first state such that $\sigma_i \neq \phi$. As σ satisfies G, it turns out that E(σ_i) = ϕ for each (EF) $\in$ G. By using the lemma 4.3, for each $n > 0$ we can build an instance J over **R** such that #const(J) $> n$ and E(J) = ϕ for each (EF) $\in$ G. Hence, the d-instance (ϕ,J) satisfies G. Let us assume that $\mathrm{D}_{\mathbf{S}}$ is an extension of Sat(**R**,G). Let (ϕ, σ_1) a d-instance over **S** such that $\sigma_1|_{\mathbf{R}}$ = J. Then # (const(σ_1) − const(σ_0)) = # const(σ_1) $\geq$ # const(J) $> n$. Hence, $\mathrm{D}_{\mathbf{S}}$ is not bounded. □.

It is clear that the consistency assumption is necessary in order to state this result. Otherwise, it suffices to consider the transaction schema T = $\{t_{id}\}$ over **R** for which we have d-gen(**R**,T) = *Empty* = Sat(**R**,G). The following example shows that positivity is also necessary:

Example 4.2 Let **R** = {P(A)} and G = {(EF)} where

$$\mathrm{E} = \sigma_{\mathrm{A} \neq a}\ \mathrm{P} \text{ and } \mathrm{F} = \sigma_{\mathrm{A} = a}\ \mathrm{P}$$

Let us consider the transaction schema (**R**,T) such that T = $\{\mathrm{i_P}, \mathrm{del_P}\}$, where:

$$\mathrm{i_P}(x) = \ \textbf{if x = a then } \mathrm{ins_P}(\mathbf{x})$$

To show that d-gen(**R**,T) = Sat(**R**,G), we will show that Sat(**R**,G) = Tr(**R**,G) = d-gen(**R**,T). In effect (**a**) If Sat(**R**,G) $\neq$ Tr(**R**,G) then there exists $\sigma \in$ Sat(**R**,G) such that $\sigma \notin$ Tr(**R**,G). Let $i \in [1 \ldots \mid \sigma \mid]$ such that E(σ_i) $\neq \phi$ and let $b \in$ E(σ_i). Then $b \neq a$. From the fact that σ satisfies G, we can conclude that there exists $j \in [0 \ldots \mid \sigma \mid]$ such that $b \in$ F(σ_j). But in this case $b = a$. Contradiction. (**b**) d-gen(**R**,T) $\subseteq$ Sat(**R**,G) (= Tr(**R**,G)) because a transaction in T can never insert a constant b in P such that $b \neq a$. Conversely, if $\sigma \in$ Tr(**R**,G) then for each $i \in [0 \ldots \mid \sigma \mid]$, we have either σ_i(P) = ϕ or σ_i(P) = $\{a\}$. Hence, for each $i \in [0 \ldots \mid \sigma \mid]$, either $\sigma_i = \mathrm{i_P}(a)(\sigma_{i-1})$ (in the case where σ_i(P) = $\{a\}$) either $\sigma_i = \mathrm{del_P}(a)(\sigma_{i-1})$ (in the case where σ_i(P) = ϕ).

We note that (**R**,G) is consistent because the d-instance (σ_0,σ_1) where σ_1(P) = $\{a\}$ satisfies G. □

4.2 Generalized Transaction Schemas

From the previous subsection we know that it is impossible to derive an operational specification having the same effect as a set of DADs through the notion of transaction schema "à la Abiteboul/Vianu". In this subsection, we generalize the notion of transaction schema. The idea for recovering multiple changes during one transition is to specify a transition as a sequence of transactions. However, as illustrated below, arbitrary sequences of transactions may violate the constraints even if each transaction is correct w.r.t. the DADs.

Example 4.3 Let us consider the transaction schema (**U**,T') of example 4.1. If we take arbitrary sequences of transactions in T' in order to update the database **U**, we may obtain a d-instance which violates UNIV. Consider the following d-instance σ:

Dσ_0		σ_1	
Prof	Stud	Prof	Stud
ϕ	ϕ	(a,b)	(a,c)

The d-instance σ violates UNIV. Note that σ_1 is obtained from σ_0 by the sequence of transactions register$'$(a,c) ; hire$'$(a,b).

Now the idea is to introduce regular expressions on transactions in order to restrict transaction sequences.

Definition 4.4 Let (**S**,T) be a transaction schema and let e be a regular expression over (the alphabet) T. Then (**S**,T,e) is a *generalized transaction schema.*

The regular language associated to the regular expression e is denoted by $\mathcal{L}(e)$. If $t \in \mathcal{L}(e)$ and $t = t_1 t_2 \ldots t_n$, a *call* to t is a transaction t', where $t' = t'_1; t'_2; \ldots ; t'_n$ and t'_i is a call to t_i, for each $i \in [1 \ldots n]$. The set of calls to transactions in $\mathcal{L}(e)$ is denoted by Call(e). The set of d-instances over **S** generated by (**S**,T,e) is defined by

d-gen(**S**,T,e) $= \{\ \sigma \in$ d-Inst(S) $\mid \forall 0 < i \leq |\sigma|, \exists\ t \in$ Call(e), $\sigma_i = t(\sigma_{i-1})\ \}$.

In this context, the notions of correctness, preservation and completeness remain the same. We say that a dynamic family $\mathrm{D}_{\mathbf{S}}$ is *g-transaction* if there exists a generalized transaction schema (**S**,T,e) such that $\mathrm{D}_{\mathbf{S}}$ = d-gen(**S**,T,e). The relationship between constraint-based specifications and transaction-based specifications can now be stated by:

> Given a DAD schema (**R**,G) is it possible to find a generalized transaction schema (**S**,T,e) such that (**S**,T,e) is correct and complete w.r.t. to (**R**,G), that is, does Sat(**R**,G) have a g-transaction extension ?

{

Example 4.4 Consider the transaction schema (**H**, T') of example 4.1 and the regular expression e_m over T' defined by $e_m = (e' + \text{hire}')^*\ (e' + \text{register}')^*$ where $e'4$ = cancel_registration + fire.

We have that d-gen(**H**,T',e_m)$|_{\mathbf{U}}$ = Sat(**U**,UNIV).

Intuitively, these transactions are correct because a call to hire will effectively hire someone only if the person concerned belongs to the historical relation S of the database i.e. has been a student in the past. It is important to note that transaction of the regular language defined by e_m does not allow one to hire and register a person "at the same time" (with the exception of someone who was already registered in the past).

As illustrated in the previous examples, auxiliary relations are necessary in order to obtain completeness. In most cases, this can be formally proved:

Proposition 4.5 Let (**R**,G) be a strong consistent DAD schema. Then there is no generalized transaction (**R**,T,e) which is correct and complete w.r.t. (**R**,G).

Proof: Let us assume (**R**,T,e) to be correct and complete w.r.t. (**R**,G). Let σ=$(\sigma_0, \ldots, \sigma_n) \in$ Sat(**R**,G) such that $\sigma \notin$ Tr(**R**,G). Then there exists a current property E appearing in G and $i \in [0 \ldots n]$ such that E$(\sigma_i) \neq \emptyset$. From the fact that σ satisfies G, it is clear that $i > 1$. The d-instance $\sigma' = (\sigma_0, \ldots, \sigma_{i-1}, \phi, \sigma_i, \ldots, \sigma_n)$ satisfies G and hence it is generated by (**R**,T,e). It turns out that the d-instance $(\sigma_0, \sigma_i, \ldots, \sigma_n)$ is generated by (**R**,T,e). Here, we have used the fact that database schema of (**R**,T,e) is **R**. Hence, $(\sigma_0, \sigma_i, \ldots, \sigma_n)$ satisfies G, which is obviously false. Contradiction. □

Instead of introducing generalized transactions schemas we could have first tried to investigate the existence of a correct transaction schemas having an effect as *close as possible* to the effect of G. Because the equality between d-gen(**S'**,T') and Sat(**R**,G) is impossible for any (**S'**,T'), a transaction schema (**S'**,T') such that d-gen(**S'**,T') generates a greatest subset of Sat(**R**,G)) may in fact present some interest. Thus, we say that a transaction schema (**S**,T) correct w.r.t. the DAD schema (**R**,G) is *maximal* if for each transaction schema (**S'**,T') correct w.r.t. (**R**,G), we have d-gen(**S'**,T')$|_{\mathbf{R}} \subseteq$ d-gen(**S**,T)$|_{\mathbf{R}}$.

The following theorem states that maximal transaction schemas do not exist for DAD schemas which can be implemented by generalized transaction schemas. We note that all subclasses of DADs considered in this paper satisfy the conditions of this theorem:

Theorem 4.6 Let (**R**,G) be a DAD schema such that Sat(**R**,G) has a g-transaction extension but no bounded extensions. Then, there exists no transaction schema (**S**,T) correct w.r.t. (**R**,G) such that (**S**,T) is maximal.

Proof: Let us assume (**S**,T) a maximal transaction schema w.r.t. (**R**,G). As Sat(**R**,G) has a g-transaction extension then Sat(**R**,G) = d-gen(**S'**,T',e)$|_{\mathbf{R}}$, for some generalized transaction schema (**S'**,T',e), such that $\mathbf{R} \subseteq$ **S'**. From the fact that Sat(**R**,G) has no bounded extension, then d-gen(**S**,T)$|_{\mathbf{R}} \neq$ Sat(**R**,G). Let $\sigma \in$ Sat(**R**,G) such that $\sigma \notin$ d-gen(**S**,T)$|_{\mathbf{R}}$. Let $\sigma' = (\sigma'_0, \sigma'_1, \ldots, \sigma'_n) \in$ d-gen(**S'**,T',e) such that $\sigma' \,|_{\mathbf{R}} = \sigma = (\sigma_0, \sigma_1, \ldots, \sigma_n)$. Let $t_1, t_2, \ldots, t_n \in$ Call(e) such that $\sigma'_{i+1} = t_{i+1}(\sigma'_i)$, for each $0 \leq i < n$. Let us consider now the transaction schema (**S"**,T") where **S"** = **S** ∪ **S'** and T"= $\{t_1, \ldots, t_n\}\cup$ T. It is clear that d-gen(**S"**,T")$|_{\mathbf{R}}$ $\subseteq$ Sat(**R**,G) and d-gen(**S"**,T")$|_{\mathbf{R}} \not\subseteq$ d-gen(**S**,T)$|_{\mathbf{R}}$, because $\sigma \in$ d-gen(**S"**,T")$|_{\mathbf{R}}$ and $\sigma \notin$ d-gen(**S**,T)$|_{\mathbf{R}}$. □

5 Elementary transaction schemas

In this section, we will build specific transaction schemas, called *elementary*. We note that in example 4.1, the transactions hire', fire, cancel-registration and register' allow us to correctly update only one tuple at a time.

In order to motivate building these elementary transaction schemas, we introduce *elementary* d-instances. A d-instance is *elementary* when changes from one state to the other is limited to an insertion or deletion of at most one tuple.

We define Sat$_1$(**R**,G) as the set of all elementary d-instances over **R** satisfying G. We note that in example 4.1, the transaction schema (**H**,T') is correct w.r.t. (**U**,Univ) and generate exactly Sat$_1$(**U**,Univ).

Historical Schemas

Without loss of generality, we assume from now on that in the DAD schema (**R**,G) each relation schema in **R** has an occurrence in G.

From proposition 4.5, we know that the first thing to do when building an elementary transaction schema for a DAD schema (**R**,G) is to enrich **R** with additional relation schemas called historical schemas. Historical schemas aim at storing information about the database changes needed in order to "check" the constraints. Thus they are induced by the structure of the DADs in G:

Definition 5.1 Let (**R**,G) be a DAD schema such that G = $\{E_1F_1,\ldots,E_nF_n\}$. For each i, let S_i be a new relation schema and X_i = $\mathrm{Tar}(F_i)$. We call $S_i(X_i)$ the *historical relation schema* associated to F_i. **H**= **R** $\cup$ $\{S_1(X_1),\ldots, S_n(X_n)\}$ is the *historical schema* associated to (**R**,G).

These historical schemas are similar to *auxiliary relations* introduced in [C]. However, their semantics are slightly different. The "updates" on relations S_i will be side effects of insertions and deletions on the initial database.

We recall that we want to show that Sat(**R**,G) has a g-transaction extension. We introduce now a special extension of Sat(**R**,G), called the *historical extension* and denote it by Hist(**R**,G). In the next section, we will show that Hist(**R**,G) is g-transaction.

Definition 5.2 Let (**R**,G) be a DAD schema, G = $\{E_1F_1,\ldots, E_nF_n\}$. Let **H** = **R** $\cup$ $\{S_1,\ldots,S_n\}$ be its historical schema where for each i, S_i is the historical relation associated to F_i. The *historical extension* of Sat(**R**,G), denoted Hist(**R**,G), is the set of d-instances $\sigma = (\sigma_0,\ldots,\sigma_m)$ $\in$ d-Inst(**H**) such that $\sigma|_{\mathbf{R}} \in$ Sat(**R**,G) and for each $i \in [0,\ldots,m]$ and $j \in [0,\ldots,n]$:

$$\sigma_i(S_j) = \bigcup_{k=0}^{i} F_j(\sigma_k)$$

In what follows, we say that a transaction t over **H** preserves the DAD schema (**R**,G) if t preserves Hist(**R**,G).

Example 5.1 Consider **R** = {P(AB), Q(BC), S(CD), R(AE)} and G = $\{E_1F_1, E_2F_2\}$ where $E_1 = \Pi_{AC}(P \bowtie Q)$, $F_1 = \Pi_{AC}(D_{A=C}(P \bowtie Q) \bowtie S)$, $E_2 = \Pi_{BD}(\Pi_{CD}(P \bowtie S) \bowtie Q)$ and $F_2 = \Pi_{BD}(D_{A=D}(R \bowtie S) \bowtie Q)$. Then the historical schema associated to G is **H** = **R** $\cup$ $\{S_1(AC),S_2(BD)\}$.

In example 3.1, the historical schema associated to the DAD schema (**U**,UNIV) is **H** = {Prof(Name,Course), Stud(Name,Address), S(Name)}.

Elementary Transaction Schema

Given a relational expression E (think of E as being the current property of a DAD EF) and an update μ over a relation Q, we are interested in the relational expression $E_\mu{}^+$ (resp. $E_\mu{}^-$) which returns the tuples inserted in (resp. deleted from) the answer to E after performing the update μ. The expressions $E_\mu{}^+$ and $E_\mu{}^-$ satisfy $E_\mu{}^+(I) = E(J) - E(I)$ and $E_\mu{}^-(I) = E(I) - E(J)$ for all I $\in$ Inst(**R**) and J=μ(I). They are defined in such a way that they do not compute E(J). The presentation of $E_\mu{}^+$ and $E_\mu{}^-$ is skipped because of space limitation (see [Am]).

Definition 5.3 The *elementary* transaction schema associated to the DAD schema (**R**,G) is (**H**,T) where

- **H** is the historical database schema associated to (**R**,G).

- T contains for each relation $\mathrm{Q} \in \mathbf{R}$ one transaction ($\mathrm{i_Q}$) for inserting a tuple in Q and one transaction ($\mathrm{d_Q}$) for deleting a tuple from Q.

These transactions are given by:

$\mathrm{i_Q}\ (u) = \textbf{if } (\mathrm{E}_1)_t^+ \subseteq [\mathrm{S}_1] \wedge \ldots \wedge (\mathrm{E}_n)_t^+ \subseteq [\mathrm{S}_n] \textbf{ then } t\ ;\ t_{\mathrm{S}_1}\ ;\ \ldots\ ;\ t_{\mathrm{S}_n} \textbf{ endif}$

$\mathrm{d_Q}\ (u) = \textbf{if } (\mathrm{E}_1)_s^+ \subseteq [\mathrm{S}_1] \wedge \ldots \wedge (\mathrm{E}_n)_s^+ \subseteq [\mathrm{S}_n] \textbf{ then } s\ ;\ s_{\mathrm{S}_1}\ ;\ \ldots\ ;\ s_{\mathrm{S}_n} \textbf{ endif}$

with $t = \mathrm{ins_Q}(u)$ and $s = \mathrm{del_Q}(u)$. The transactions t_{S_i} (resp. s_{S_i}) insert in the historical schemas the new tuples that answer the query F_i after the execution of t (resp. s). They are defined by:

$t_{\mathrm{S}_i} = \textbf{while } v \in (\mathrm{F}_i)_t^+ \textbf{ do } \mathrm{ins}_{\mathrm{S}_i}(v) \textbf{ done}.$

$s_{\mathrm{S}_i} = \textbf{while } v \in (\mathrm{F}_i)_s^+ \textbf{ do } \mathrm{ins}_{\mathrm{S}_i}(v) \textbf{ done}.$

The transactions hire', register', fire and cancel_registration of example 4.1 are the simplified versions of the elementary transactions $\mathrm{i_{Prof}}$, $\mathrm{i_{Stud}}$, $\mathrm{d_{Prof}}$ and $\mathrm{d_{Stud}}$ respectively.

Theorem 5.1 Let ($\mathbf{R}$,G) be a DAD schema and ($\mathbf{H}$,T) its associated elementary transaction schema. Then ($\mathbf{H}$,T) preserves Hist($\mathbf{R}$,G) and d-gen($\mathbf{H}$,T) is an extension of Sat_1($\mathbf{R}$,G).

Proof: Let $\mathrm{G} = \{(\mathrm{E}_1\mathrm{F}_1),\ldots,(\mathrm{E}_m\mathrm{F}_m)\}$. First, we show that ($\mathbf{H}$,T) preserves Hist($\mathbf{R}$,G). Let $\sigma = (\sigma_0, \ldots, \sigma_n) \in \mathrm{Hist}(\mathbf{R},\mathrm{G})$. We know that $\sigma\mid_{\mathbf{R}} \in \mathrm{Sat}(\mathbf{R},\mathrm{G})$ and $\sigma_i(\mathrm{S}_j) = \bigcup_{k=0}^{i} \mathrm{F}_j(\sigma_k)$, for each $j \in [1, \ldots, m]$. Let $\sigma_{n+1} = t(\sigma_n)$, where $t \in \mathrm{Call(T)}$. It is clear that $\sigma_{n+1}(\mathrm{S}_j) = \bigcup_{k=0}^{n+1} \mathrm{F}_j(\sigma_k)$, for each $j \in [1, \ldots, m]$. Let us assume that $(\sigma, \sigma_{n+1})|_{\mathbf{R}}$ does not satisfy a DAD (EF) in G. From the fact that $\sigma|_{\mathbf{R}}$ satisfies (EF), it turns out that there exists $u \in \mathrm{E}(\sigma_{n+1})$ such that $u \notin \mathrm{F}(\sigma_i)$ for each $i \in [0 \ldots n]$ and $u \notin \mathrm{E}(\sigma_n)$. Hence, $u \in \mathrm{E}(\sigma_{n+1}) - \mathrm{E}(\sigma_n) = \mathrm{E}_t{}^+(\sigma_n)$. However, from the fact that $u \notin \mathrm{F}(\sigma_i)$, for each $i \in [0 \ldots n]$ we can conclude that $u \notin \sigma_i(\mathrm{S})$, where S is the historical relation associated to F. Hence, $t = t_{id}$ and so $u \notin \mathrm{E}(\sigma_{n+1})$. Contradiction.

Now we show that d-gen($\mathbf{H}$,T)$|_{\mathbf{R}} \subseteq \mathrm{Sat}_1(\mathbf{R},\mathrm{G})$. It suffices to show that the d-instances in d-gen($\mathbf{H}$,T)$|_{\mathbf{R}}$ are elementary. Let $i \in [0 \ldots n-1]$. Then $\sigma_{i+1} = t(\sigma_i)$ for $t \in \mathrm{Call(T)}$.

(**a**) If $t = \mathrm{i_P}(u)$ then either $\sigma_{i+1} = \sigma_i$ either

$$\begin{aligned} \sigma_{i+1}(\mathrm{Q}) &= \sigma_i(\mathrm{Q}) \text{ if } \mathrm{Q} \neq \mathrm{P} \text{ and} \\ \sigma_{i+1}(\mathrm{P}) &= \sigma_i(\mathrm{P}) \cup \{u\} \end{aligned}$$

(**b**) If $t = \mathrm{d_P}(u)$ then either $\sigma_{i+1} = \sigma_i$ either

$$\begin{aligned} \sigma_{i+1}(\mathrm{Q}) &= \sigma_i(\mathrm{Q}) \text{ if } \mathrm{Q} \neq \mathrm{P} \text{ and} \\ \sigma_{i+1}(\mathrm{P}) &= \sigma_i(\mathrm{P}) - \{u\} \end{aligned}$$

In both cases (**a**) and (**b**)

$$\Sigma_{\mathrm{R}\in\mathbf{R}} \mid \sigma_{i+1}(\mathrm{R}) - \sigma_i(\mathrm{R}) \mid + \mid \sigma_i(\mathrm{R}) - \sigma_{i+1}(\mathrm{R}) \mid \leq 1$$

Hence, σ is elementary.

Now, we show that $\mathrm{Sat}_1(\mathbf{R},\mathrm{G}) \subseteq$ d-gen($\mathbf{H}$,T)$|_{\mathbf{R}}$. Let $\sigma = (\sigma_0, \ldots, \sigma_n) \in \mathrm{Sat}_1(\mathbf{R},\mathrm{G})$ and $i \in [0 \ldots n-1]$. Let $\mathrm{I} = \sigma_i$ and $\mathrm{J} = \sigma_{i+1}$. As σ is elementary then

$$\Sigma_{\mathrm{R}\in\mathbf{R}} \mid \mathrm{J(R)} - \mathrm{I(R)} \mid + \mid \mathrm{I(R)} - \mathrm{J(R)} \mid \leq 1$$

Hence, either I = J either there exists $\mathrm{R} \in \mathbf{R}$ such that $\mid \mathrm{J(R)} - \mathrm{I(R)} \mid = 1$ or $\mid \mathrm{I(R)} - \mathrm{J(R)} \mid = 1$. Let $\sigma' = (\sigma'_0, \ldots, \sigma'_n) \in \mathrm{Hist}(\mathbf{R},\mathrm{G})$ such that $\sigma'|_{\mathbf{R}} = \sigma$ and I' $= \sigma'_i$ and J' $= \sigma'_{i+1}$. If $\mid \mathrm{J(R)} - \mathrm{I(R)} \mid = 1$, let $\{u\} = \mathrm{J(R)} - \mathrm{I(R)}$. From the fact that σ satisfies G, it is not difficult to see that J' $= \mathrm{i_R}(u)$(I') $= \mathrm{ins_R}(u)$(I'). If $\mid \mathrm{I(R)} - \mathrm{J(R)} \mid = 1$ then a similar argument can be used in order to conclude that J' $= \mathrm{d_R}(v)$(I') $= \mathrm{del_R}(v)$(I'), where $\{v\} = \mathrm{I(R)} - \mathrm{J(R)}$. □

6 Generalized Transaction Schemas

The generalized transaction schema (**H**,T,e) built next generates exactly Hist(**R**,G) and its definition uses the elementary transactions defined in the previous section. This is done only for a subclass of DADs. The following example illustrates the difficulty to manage any DAD.

Example 6.1 Consider **R** = {P(ABC),Q(CDE),R(ABDE)} and G = {EF} where

$$\mathrm{E} = \Pi_{\mathrm{ABD}}(\Pi_{\mathrm{DE}}\mathrm{Q} \bowtie \Pi_{\mathrm{ABE}}(\mathrm{P} \bowtie \mathrm{R})) \text{ and } \mathrm{F} = \Pi_{\mathrm{ABD}}(\Pi_{\mathrm{AB}}\mathrm{P} \bowtie \Pi_{\mathrm{CD}}(\mathrm{Q} \bowtie \mathrm{R})).$$

First, we know that in order to have correctness and completeness, preservation of Hist(**R**,G) should hold. Second, for the sake of completeness, we must allow one several calls of an elementary transaction to make transition between states. We know that $\mathrm{i_R}$ preserves Hist(**R**,G). However $\mathrm{i_R};\mathrm{i_R}$ does not preserve Hist(**R**,G). To show that, consider the following table. Here, S stands for the historical schema associated to the past property F. The answers of the queries E and F have been added in the table for the purpose of the example.

	P	Q	R	S	E	F
σ_0	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ
σ_1	(a_1, b_1, c_1)	(c_1, d_1, e_1)	(a_2, b_1, d_1, e_1)	(a_1, b_1, d_1)	ϕ	(a_1, b_1, d_1)
σ_2	(a_1, b_1, c_1) (a_2, b_2, c_2)	(c_2, d_1, e_1)	ϕ	(a_1, b_1, d_1)	ϕ	ϕ
σ'_2	(a_1, b_1, c_1) (a_2, b_2, c_2)	(c_2, d_1, e_1)	(a_1, b_1, d_1, e_1)	(a_1, b_1, d_1) (a_2, b_2, d_1)	(a_1, b_1, d_1)	(a_1, b_1, d_1) (a_2, b_2, d_1)
σ_3	(a_1, b_1, c_1) (a_2, b_2, c_2)	(c_2, d_1, e_1)	(a_1, b_1, d_1, e_1) (a_2, b_2, d_1, e_1)	(a_1, b_1, d_1) (a_2, b_2, d_1)	(a_1, b_1, d_1) (a_2, b_2, d_1)	(a_1, b_1, d_1) (a_2, b_2, d_1)

The d-instance $(\sigma_0, \sigma_1, \sigma_2) \in$ Hist(**R**,G). The instance σ'_2 is the result of applying $\mathrm{i_R}(a_1, b_1, d_1, e_1)$ on state σ_2 and the instance σ_3 is the result of applying $\mathrm{i_R}(a_2, b_2, d_1, e_1)$ on state σ'_2. We can see that the d-instance $(\sigma_0, \sigma_1, \sigma_2, \sigma_3) \notin$ Hist(**R**,G) because its restriction on **R** does not satisfy G. This shows that $\mathrm{i_R};\mathrm{i_R}$ does not preserve Hist(**R**,G).

The problem here is that we cannot "separate" elementary transactions which insert into the answer of E from those which insert into the answer of F. We note that this "separation" can be done in an obvious manner for the *University* example (see example 4.4).

Our first step is to exhibit sufficient conditions which ensure that if each transaction of a sequence s preserves Hist(**R**,G) then a call to s also preserves Hist(**R**,G).

Definition 6.1 Let (**R**,G) be a DAD schema and E be a relational expression over **R**. Let **H** be the historical schema associated to (**R**,G). E is said to be *dynamically decreasing* (d-decreasing) w.r.t. a transaction t over **H** if for each d-instance $(\sigma_0,\ldots,\sigma_n)$ in Hist(**R**,G) we have

$$\mathrm{E}(\sigma_{n+1}) \subseteq \bigcup_{i=0}^{n} \mathrm{E}(\sigma_i), \text{ if } \sigma_{n+1} = t(\sigma_n).$$

We say that E is *dynamically decreasing* (d-decreasing) w.r.t. a p-transaction t, if E is d-decreasing w.r.t. each call to t.

Note that if E is d-decreasing w.r.t. t and s then it is also d-decreasing w.r.t. $t; s$.

Theorem 6.1 Checking dynamic decrease is undecidable.

Proof: Let us consider the following problem

(**DEC**) Given a relational expression E and a p-transaction t, is it true that E(J) $\subseteq$ E(I) for all instances I,J $\in$ Inst(**R**) such that J = t(I) ?

We show that if checking dynamic decrease is decidable then (**DEC**) is also decidable. Indeed, let us assume that E is d-decreasing w.r.t. t and let I,J $\in$ Inst(**R**) such that J $= t$(I). Let us consider the d-instance (ϕ,I,J). From the fact that E is d-decreasing w.r.t. t, then E(J) $\subseteq$ E(I) $\cup$ E(ϕ) $=$ E(I). Hence, the problem (**DEC**) has a positive answer. Let us assume that E is not d-decreasing w.r.t. t. Then there exists a d-instance $(\sigma_0, \ldots, \sigma_n)$ and $t' \in$ Call(t) such that $\sigma_{n+1} = t'(\sigma_n)$ and $\mathrm{E}(\sigma_{n+1}) \not\subseteq \bigcup_{i=0}^{n} \mathrm{E}(\sigma_i)$. Hence, $\mathrm{E}(\sigma_{n+1}) \not\subseteq \mathrm{E}(\sigma_n)$. Then, the problem (**DEC**) has a negative answer.

Let E be a relational expression and R a relation schema which does not appear in E. Let us consider the relational expression F $=$ E $\times$ R and $t = \mathrm{ins}_{\mathrm{R}}$. As R does not appear in E, it is clear that for each constant tuple u over R and for each instance I over R, if J $= t(u)$(I) then :

$$\mathrm{F(J)} - \mathrm{F(I)} = \begin{cases} \mathrm{E(I)} \times (u) & \text{if } u \notin \mathrm{I(R)} \\ \phi & \text{otherwise} \end{cases}$$

Hence, F is d-decreasing w.r.t. t iff, for each constant tuple u over R, E $\times$ $(u) = \Theta$. But this is true iff E $\equiv \Theta$. It turns out that, the decidability of (**DEC**) entails the decidability of the equivalence between two relational expressions. It is well known that this problem is undecidable ([Ka]). □

Lemma 6.2 Let (**R**,G) be a DAD schema and **H** its historical schema. Let t and s be transactions over **H** preserving Hist(**R**,G). If for each EF in G, F is d-decreasing w.r.t. t or E is d-decreasing w.r.t. s, then $t;s$ preserves Hist(**R**,G). It turns out that if E or F are d-decreasing w.r.t. t then t^* preserves Hist(**R**,G), where t^* stands for a sequence of calls to t.

Proof: Let EF $\in$ G and let us assume that either F is d-decreasing w.r.t. t either E is d-decreasing w.r.t. s. Let $\sigma = (\sigma_0, \ldots, \sigma_n)$ a d-instance satisfying EF and $\sigma_{n+1} = (t';s')(\sigma_n)$ where t' and s' are respectively calls to t and s. Let $\tau_n = t'(\sigma_n)$. From the fact that t preserves EF, it turns out that (σ, τ_n) satisfies EF. From the fact that s preserves EF, it turns out that $(\sigma, \tau_n, \sigma_{n+1})$ satisfies EF. Let us assume that (σ, σ_{n+1}) does not satisfy EF. As σ satisfies EF, we can conclude that there exists $u \in \mathrm{E}(\sigma_{n+1})$ such that $u \notin \mathrm{F}(\sigma_i)$, for each $i \in [0 \ldots n]$. We have two possibilities : (**a**) if F is d-decreasing w.r.t. to t then $u \notin \mathrm{F}(\tau_n)$. In this case, we conclude that $(\sigma, \tau_n, \sigma_{n+1})$ does not satisfy EF. Contradiction. (**b**) If E is d-decreasing w.r.t. s then either $u \in \mathrm{E}(\tau_n)$ either there exists $i \in [0 \ldots n]$ such that $u \in \mathrm{E}(\sigma_i)$. In this case, we have that (σ, τ_n) does not satisfy EF. Contradiction. □

This result can be directly used to show that any call to t in the language of the expression e_m of the *University* Example (see 4.4) preserves the DAD schema (**U**,UNIV). On the other hand, if we consider the DAD schema of example 6.1 we can easily see that neither E nor F are d-decreasing w.r.t. transactions $\mathrm{i_R}$, $\mathrm{i_P}$ and $\mathrm{i_Q}$.

The problem to deal with general DADs arises when a relational schema P occurs in both the current property (E) and the past property (F). In this case, E and F may not be d-decreasing w.r.t. to the elementary transactions $\mathrm{i_P}$ and $\mathrm{d_P}$. For that reason, we will restrict our attention to a subclass of DADs, called *regular*. For a dependency EF in this class, we are able to "separate" transactions which insert into the answers of E from those which insert into the answer of F.

Definition 6.2 A DAD EF is *regular* when E is a SPJRU query, F is a SPRJ query and the set of relations occurring in F is not included in the set of relations of E (sch(F) $\not\subseteq$ sch(E)). Note that this class contains DIDs of the form RS(X) such that R$\neq$S.

For instance, the DAD presented in example 3.2 is regular while the one presented in example 3.3 is not.

If sch(F) $\cap$ sch(E) is empty then there is nothing to do in order to separate transactions inserting on E from those inserting on F. Note that if EF is regular, sch(E) $\cap$ sch(F) may not be empty. Thus, we need to introduce some technical modifications on the elementary transactions in order to accomplish the separation in that case. Let ($\mathbf{R}$,G) be a DAD schema with G = $\{E_1F_1,\ldots,E_nF_n\}$:

- arbitrary choose a relation schema $P_i \in$ sch(F_i) $-$ sch(E_i).
- for each R $\in$ $\mathbf{R}$ consider the set $\Lambda_R = \{i \mid 0 \leq i \leq n$ and R $\in$ sch(E_i)$\cap$sch(F_i)$\}$. Let us assume Λ_R=$\{i_1,\ldots,i_k\}$. We define the p-transaction $t_R(\vec{x})$ = $\mathrm{erase}_{P_{i_1}};\ldots;\mathrm{erase}_{P_{i_k}};\mathrm{i}_R(\vec{x})$, where $\mathrm{i}_R(\vec{x})$ is the elementary transaction inserting on R.

Now we work with the modified transaction schema ($\mathbf{H}$,T) where T = $\{t_R, d_R \mid R \in \mathbf{R}\}$ (the transactions d_R have not been modified).

Example 6.2 Consider the DAD schema of example 5.1. Let us choose S $\in$ sch(F_1)$-$sch(E_1) and R $\in$ sch(F_2)$-$sch(E_2). Then we have

- Λ_P=$\{1\}$, Λ_Q=$\{1,2\}$, Λ_S = $\{2\}$, Λ_R=ϕ.
- t_P = erase_S;i_P, t_Q = erase_S; erase_R;i_Q, t_S = erase_R;i_S and t_R = i_R.

Lemma 6.3 For each P $\in$ $\mathbf{R}$ and for each EF $\in$ G :

1. E and F are d-decreasing w.r.t. d_P.
2. If P $\notin$ sch(E) then E is d-decreasing w.r.t. t_P.
3. If P $\in$ sch(E) then F is d-decreasing w.r.t. t_P.
4. t_P preserves EF.

Proof:
(1): Immediate from the fact that neither E nor F contain the difference operator.
(2): If P $\notin$ sch(E) then i_P has no effect on the answer of E. As E does not contain the difference operator then E is d-decreasing w.r.t. transactions erase_R.
(3): If P $\notin$ sch(F) then the result is straightforward. If P $\in$ sch(E) $\cap$ sch(F) then $t_P = \ldots;$ erase_S ; $\ldots$; i_P for S $\in$ sch(F) $-$ sch(E). Hence, after the execution of t_P, the set of answers of F is empty.
(4): either $t_P = \mathrm{i}_P$ either $t_P = s$; i_P where $s = \ldots;\mathrm{erase}_S;\ldots$. From the facts that s and i_P preserve EF and that F is d-decreasing w.r.t. s we have that t_P preserves EF (see lemma 6.2). □

In order to exhibit the regular expression e such that the generalized transaction schema ($\mathbf{H}$,T,e) has the same effect as the regular DAD schema ($\mathbf{R}$,G), we use the graph (Node,Edge), improperly called G, where Node = $\mathbf{R}$ and (P,Q) $\in$ Edge if and only if there exists EF $\in$ G such that P $\in$ sch(E) and Q $\in$ sch(F)$-$sch(E).

For instance, the graph associated with the DAD schema of example 5.1 is given in figure 2.

Next we only consider acyclic graph G.

The partition N_0, ..., N_k associated with the topological sorting of G is defined by:

N_0 is the set of nodes in G having no incident edges,

N_{i+1} is the set of nodes in the graph obtained by removing the nodes in $\bigcup_{j=0..i}N_j$ from G and having no incident edges.

Example 6.3 For the graph G of figure 2, N_0 =$\{P,Q\}$, N_1 =$\{S\}$ and N_2=$\{R\}$.

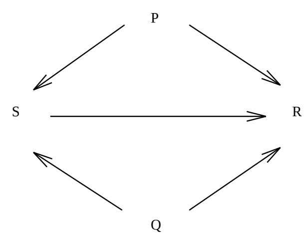

Figure 2: The graph G

Intuitively, the topological partition of G is the basis for ordering the transactions t_Q inserting tuples. It is not difficult to see that if the graph (Node,Edge) is acyclic then Node $= \bigcup_i^n \mathrm{N}_i$.

Definition 6.3 Let ($\mathbf{R}$, G) be an acyclic regular DAD schema and let ($\mathbf{H}$,T) be the slightly modified elementary transaction schema defined above. Let N_0, ..., N_k be the topological partition of G. The *regular expression* associated to ($\mathbf{R}$, G) is defined by:

$$e = (\sum_{\mathrm{P} \in \mathrm{N_1}} t_\mathrm{P} + \sum_{\mathrm{P} \in \mathbf{R}} \mathrm{d_P})^* \ldots (\sum_{\mathrm{P} \in \mathrm{N_k}} t_\mathrm{P} + \sum_{\mathrm{P} \in \mathbf{R}} \mathrm{d_P})^*$$

Example 6.4 The regular expression associated to the DAD schema of example 5.1 is the following:

$$e = (t_\mathrm{P} + t_\mathrm{Q} + \mathrm{d_P} + \mathrm{d_Q} + \mathrm{d_S} + \mathrm{d_R})^*(t_\mathrm{S} + \mathrm{d_P} + \mathrm{d_Q} + \mathrm{d_S} + \mathrm{d_R})^*(t_\mathrm{R} + \mathrm{d_P} + \mathrm{d_Q} + \mathrm{d_S} + \mathrm{d_R})^*$$

In the *University* Example (see example 4.4), e_m is the regular expression associated to the DAD schema UNIV.

We conclude by stating the main result of this section:

Theorem 6.4 Let ($\mathbf{R}$, G) be an acyclic regular DAD schema. Then, the generalized transaction schema ($\mathbf{H}$,T,e) where $\mathbf{H}$ is the historical schema, T is the modified elementary transaction schema and e is the regular expression associated to ($\mathbf{R}$, G), is correct and complete w.r.t. the DAD schema ($\mathbf{R}$, G), that is d-gen($\mathbf{H}$,T,e) = Hist($\mathbf{R}$,G).

Proof: Let us show that ($\mathbf{H}$,T,e) preserves Hist($\mathbf{R}$,G).
Let $e = e_1{}^*;\ldots;e_k{}^*$ where $e_i = (\sum_{\mathrm{P} \in \mathrm{N_i}} t_\mathrm{P} + \sum_{\mathrm{P} \in \mathbf{R}} \mathrm{d_P})$.

It is clear that $k > 1$. Indeed, from the fact that the graph (Node,Edge) associated to ($\mathbf{R}$, G) is acyclic we can show that $k = 1$ entails N_1 = Node = $\mathbf{R}$. However, if (P,Q) $\in$ Edge then Q $\notin \mathrm{N}_1$.

Let EF $\in$ G. We will use lemma 6.2 to show that e preserves EF. From the fact that the graph (Node,Edge) is acyclic, we can conclude that

$$\mathrm{N}_1 \cup \ldots \cup \mathrm{N}_k = \mathrm{Node} = \mathbf{R}$$

Hence, there exists $i \in [1 \ldots k]$ such that $\mathrm{N}_i \cap$ (sch(F) $-$ sch(E)) $\neq \emptyset$ (1).

Let m be smallest integer such that (1) holds. We have that $m > 1$. Indeed, if Q $\in \mathrm{N}_1$ then for each P $\in \mathbf{R}$ we have (P,Q) $\notin$ Edge. Hence Q $\notin$ sch(F) $-$ sch(E).

We show now that:

(**1**): for each $i \in [1 \ldots m-1]$, e_i preserves G and F is d-decreasing w.r.t. e_i. Hence, $e_1{}^*;\ldots;e_{m-1}{}^*$ preserves G and F is d-decreasing w.r.t. $e_1{}^*;\ldots;e_{m-1}{}^*$. Indeed, let $t \in$ Call(e_i). If $t = \mathrm{d_P}(u)$, then F is d-decreasing w.r.t. t. Let $t = t_\mathrm{P}(u)$. As P $\in \mathrm{N}_i$ and $i < m$, then P $\notin$ sch(F)

– sch(E). Hence, either P $\notin$ sch(F) either P $\in$ sch(F) $\cap$ sch(E). In both cases, F is d-decreasing w.r.t. t_{P}.

(**2**): for each $i \in [m \ldots k]$, e_i preserves G and E is d-decreasing w.r.t. e_i. Hence, $e_m{}^*;\ldots;e_k{}^*$ preserves G and E is d-decreasing w.r.t. $e_m{}^*;\ldots;e_k{}^*$. Indeed, let $t \in$ Call(e_i). If $t = \mathrm{d_P}(u)$, then E is d-decreasing w.r.t. t. Let $t = t_{\mathrm{P}}(u)$, P $\in$ N$_i$. Because N$_m$ $\cap$ sch(F) – sch(E) $\neq \emptyset$, then sch(E) $\subseteq$ N$_1$ $\cup \ldots \cup$ N$_{m-1}$. From the fact that the graph (Node,Edge) is acyclic and $i > m$, it turns out that sch(E) $\cap$ N$_i$ = $\emptyset$. Hence, E is d-decreasing w.r.t. t_{P}.

Thus, from lemma 6.2, e preserves G.

Let us show now that (**H**,T,e) is complete w.r.t. (**R**,G). We will show that Hist(**R**,G) $\subseteq$ d-gen(**H**,T,e). Let $\sigma = (\sigma_0, \ldots, \sigma_n) \in$ Hist(**R**,G) and I = σ_i, J = σ_{i+1}, $i \in [0 \ldots n-1]$. For each P $\in$ **R**, consider the following set of tuples :

$$\begin{aligned} \mathrm{I(P)} &= \{w_1, \ldots, w_p\} \\ \mathrm{In(P)} &= \mathrm{J(P)} - \mathrm{I(P)} = \{u_1, \ldots, u_m\} \\ \mathrm{Out(P)} &= \mathrm{I(P)} - \mathrm{J(P)} = \{v_1, \ldots, v_l\} \end{aligned}$$

and the following transactions :

$$\begin{aligned} \mathrm{trans\_In(P)} &= \mathrm{t_P}(u_1);\ldots;\ \mathrm{t_P}(u_m) \\ \mathrm{trans\_Out(P)} &= \mathrm{d_P}(v_1);\ldots;\ \mathrm{d_P}(v_l) \\ \mathrm{restore(P)} &= \mathrm{t_P}(w_1);\ldots;\ \mathrm{t_P}(w_p) \end{aligned}$$

For each N$_i$ = {P$_1$,...,P$_{k_i}$}, let

$$\begin{aligned} \mathrm{trans\_In(N}_i) &= \mathrm{trans\_In(P_1)};\ \ldots\ ;\ \mathrm{trans\_In(P}_{k_i}) \\ \mathrm{restore(N}_i) &= \mathrm{restore(P_1)};\ \ldots\ ;\ \mathrm{restore(P}_{k_i}) \end{aligned}$$

and trans_Out = trans_Out(R$_1$) ; ... trans_Out(R$_p$) where **R** = {R$_1$,...,R$_p$}.

Consider the following transaction $t \in$ Call(e) :

$$\begin{aligned} t \ = \ & \mathrm{trans\_In(N_1)}\ ;\ \mathrm{trans\_In(N_2)}\ ;\ \mathrm{restore(N_2)}\ ;\ \ldots\ ; \\ & \mathrm{trans\_In(N}_k)\ ;\ \mathrm{restore(N}_k)\ ;\ \mathrm{trans\_Out} \end{aligned}$$

We show that the transactions restore(N$_i$) do not erase the relations appearing in N$_1$ $\cup \ldots \cup$ N$_i$. Indeed, let t_{P} be a transaction in the sequence constituting the transaction restore(N$_i$), where P $\in$ N$_i$ and let us assume that there is a transaction composing t_{P} which erases a relation R appearing in N$_j$, $1 \leq j \leq i$. From the definition of t_{P}, there exists EF $\in$ G such that P $\in$ sch(E) $\cap$ sch(F) and R $\in$ sch(F) – sch(E). From the definition of the graph (Node,Edge), it turns out that (P,R) is an edge. However, the topological partition of the acyclic graph (Node,Edge) entails that if (P,R) is an edge, P $\in$ N$_i$ and R $\in$ N$_j$ then $i < j$. Contradiction.

It can be easily checked that the transactions trans_Out have the desired effect from the fact that σ satisfies G. Thus, if J' = t(I) then it is clear that J'$_{\mathbf{R}}$ = J$_{\mathbf{R}}$.

For each EF $\in$ G, let S be the historical relation schema For each EF $\in$ G, let S be the historical relation associated to F. We have that J'(S) = I(S) $\cup$ F(J') = I(S) $\cup$ F(J). Hence, J' = J. Note that the acyclicity of the graph (Node,Edge) is used to conclude that all insertions and deletions over the relations in **R** appear in the transaction t. □

7 Related Work

7.1 Comparison with active databases

Active database is a new promising technology [AHV]. Several active database languages and models have been proposed, several prototypes and commercial systems have been developed

[WC]. Fundamental work is also emerging [AWH] [BM] [PV]. The active database paradigm provides DBMS with a new component called "trigger system". The trigger system supports automatic triggering of actions, usually updates, in response to events which are in general updates too. Languages designed to express triggers are based on ECA (Event-Condition-Action) rule. Active databases have been widely investigated for implementing integrity maintenance and seems to provide a well-suited framework for developing repair techniques. Given integrity constraints, active rules are generated in order to repair constraint violation.

Thus, the main difference between the present work and active database management of integrity is the overall approach : our approach is transaction-based (as defined in the introduction) whereas the active database framework supports repairing techniques and focuses mostly on static constraint.

We continue the discussion on repair technique, termination of rule execution, automatic generation of rules and completeness.

- *Repair technique*

  On the one hand, the subclass of dynamic constraints considered in the present paper does not lend itself to repair techniques. Indeed, let us consider our running toy example: `in order to be a professor one should have been a student in the (strict) past`. Assume that the insertion of M. King in the relation Prof is detected by the trigger system and that the condition `M. King has been a student` fails. Then the insertion would violate the constraint and it is clear that there is no way to repair the past. Thus, for this type of constraints, a transaction-based approach seems more appropriate. For our example, it provides the application programmer with a transaction (and a programming discipline) which, roughly speaking, checks if the past history of the database allows one to insert M. King in the relation Prof before performing the insertion. Rollback processing is never used in our framework.

  On the other hand, it should be noticed that the transaction-based approach does not preclude repair actions, in general. [Am] and [AB2] consider other types of dynamic constraints e.g. dynamic functional dependencies [V] and among the transactions generated, some are including repairing actions. Let us try to give an example without being too technical. Consider the following constraint which is very close to our running example and called a non strict DID: `in order to be a TA one should have been a student or be a student now`. Among the transactions dedicated to perform insertion in the relation TA, one transaction will check if the person to hire as a TA was a student in the past and if it is not the case, the transaction will insert the person both in the relations TA and Student at the same time.

- *Termination, Conflicting updates, Rollback* :

  The semantics of trigger systems is generally defined by means of execution models and the behavior of complex active rule sets is very difficult to predict. The main problems arising are : termination, confluence and also conflicting updates. In our framework, we are faced with none of these problems and proving that a generalized transaction schema has the same effect as a set of DADs do not make use of any assumption concerning termination. Constraint management using active database is in general coupled with analysis methods to determine potential infinite rule firing sequences and the designer is then responsible for preventing non termination. Concerning conflicting updates, in general it is not guaranteed that the action of a rule fired to repair a constraint violation does not violate another constraint. In this case the user program is rolled back. Work is in progress concerning termination, conflicting updates, preventing rollback (see for instance [CFPT]).

- *Automatic generation of active rules* :

  Given a set of constraints, the problem of automatically generating active rules to enforce the constraints has been extensively studied [WC] [FP]. In general, there exists many ways to repair a constraint violation. The choice of the rules as well as the analysis of the set of rules with respect to termination and conflict are partially automated and the designer is responsible of strategic decisions. In our framework, generation of transactions and also of regular expressions are fully automated.

- *Completeness* :

  As said before, a set of alternative rules may be used to repair constraint violation. Most proposals have opted for deterministic repair and require that one rule be chosen. The work of [CFPT] is among the exceptions. The choice of one rule among the alternative rules entails that the database is not a strict representation of the real world's behavior characterized by the constraint. This may be penalizing for some applications.

In conclusion, we believe that the active database approach and the transaction-based approach are complementary. The transaction-based approach is not universal although we are investigating other dynamic constraints than DADs and dynamic functional dependencies. From a practical point of view our approach is probably less modular than active database.

7.2 Comparison with transaction logic programming

[BK] introduces a very elegant and powerful transaction language based on a formally well-defined logic called transaction logic. Transaction logic has a natural model theory and a sound and complete proof theory. Moreover it allows users to program transactions. The language proposed is very rich and provides features like committed updates, bulk updates, hypothetical reasoning and dynamic constraints on transaction execution.

The transaction logic language (abbreviated $\mathcal{T}_{\mathcal{R}}$) of [BK] is a language allowing one to write update programs and queries like SdetTL [AV2] does. However they differ in many aspects and an incomplete list of these differences includes: $\mathcal{T}_{\mathcal{R}}$ is logic-based whereas SdetTL is procedural; $\mathcal{T}_{\mathcal{R}}$ applies to arbitrary first-order formulas and thus has a wide application domain (AI, database, object oriented database, etc) whereas SdetTL focuses on relational database only.

As stressed in [BK] (section 6.1), procedures that handle the details of consistency maintenance are easily defined in $\mathcal{T}_{\mathcal{R}}$. However, we insist here on the fact that $\mathcal{T}_{\mathcal{R}}$ does not allow the designer to specify the constraints in first order logic and to expect the constraints to be maintained directly from this specification. The user has to write $\mathcal{T}_{\mathcal{R}}$ procedures to maintain the constraints. Given a $\mathcal{T}_{\mathcal{R}}$ transaction and a constraint, $\mathcal{T}_{\mathcal{R}}$ supports constraint verification through hypothetical reasoning in the sense that it can be checked whether every execution of the transaction will satisfy the constraint. No such support is provided by SdetTL. It should be mentioned too that $\mathcal{T}_{\mathcal{R}}$ allows one to specify in a very easy way constraints on execution paths of transaction. These *dynamic* constraints are out of the scope of our study (although one could see our regular expressions on transactions as instances of such constraints). Planning system for robot navigation provides examples of application for these constraints.

In conclusion $\mathcal{T}_{\mathcal{R}}$ is a very appealing transaction language and subsumes SdetTL. Investigating the relationship between constraint schemas and transaction schemas defined using $\mathcal{T}_{\mathcal{R}}$ as its transaction language component remains open.

7.3 Other comparisons

Recently, particular interest has been dedicated to the study of the relationship between declarative and operational database specifications.

The pioneer work of Abiteboul and Vianu [AV2, AV3] is of course the closest to our work. They have been the first to investigate in this way the idea of encapsulation. Encapsulation consists in restricting the updates to a predefined set of valid updates. Moreover, these updates are operationally defined through a transaction language. The framework investigated in [AV2, AV3] to enforce static constraints does not extend directly for the dynamic integrity maintenance, as we show in this paper.

The work presented in [CCF] is motivated by solving the same problem as the one discussed in this paper. However the update encapsulation is realized differently by means of pre-post conditions. These conditions are declaratively specified in an extended temporal language. Thus transactions are not defined in an operational way. The declarative level and the operational level tend to collapse. The results are not constructive either and the notion of completeness particularly emphasized in our approach, is totally absent. The comparison of [Li, FS, BR] with our work leads to a similar conclusion.

We would like to point out here that the work of [STSW] is an original contribution to a slightly different problem. Their approach belongs to the repair technic in the sense that, given a static or transition constraints and given a user program/transaction, they propose to refine the user program by rewriting methods updating objects in order to obtain a program which enforces the constraint. This approach is very much interesting although it is not always possible to derive a *consistent specialization of the original operations*. It seems also that the method hardly applies in general to multiple constraints.

[C, LS] do not address the issue of operational specification. They both develop a "history-less" approach for temporal constraint checking. However their work has influenced ours as already explain in the paper: the historical database schemas introduced in this paper are almost identical to the *extended database states* defined by J. Chomicki in [C].

In the framework of object-oriented database, the work of [Su] on object migrations focuses also on the relationship between declarative and operational database specifications. As in our approach, the operational part is given by transaction schemas built up from update programs. The main difference with our approach resides in the constraints considered. The scope of the constraints investigated in [Su] is the migration of objects from one class to another. Thus constraints called *migration inventories* are defined by means of *rolesets* which are sets of sequences of classes. In this framework, simple transaction schemas suffice to "implement" migration inventories and no generalization of transaction schema is required as in our framework. However note that these constraints are quite different from DADs.

8 Conclusions and further work

In this paper we have investigated the relationship between declarative and operational specifications of dynamic databases. We have introduced a class of dynamic constraints, the Dynamic Algebraic Dependencies (DADs) for which the classical problems of consistency and implication are undecidable. We have introduced a generalized notion of transaction schema based on a regular language over an alphabet of transactions. For a significant subclass of these constraints, we have given an operational equivalent specification based on generalized transaction schemas. This result is of practical interest because it can be used in order to provide the database application programmer with primitive update programs and programming rules knowing that they are sufficient for writing any "good" update program. One open direction of research is to design the right tools for the programmer.

Clearly, it is important also to investigate other classes of dynamic dependencies. We already have obtained positive results [AB2] concerning the operational specification of Vianu's Dynamic Functional Dependencies (DFDs) [V] using generalized transaction schemas. For both DADs and DFDs, the regular expressions introduced in transaction schemas are used in a very loose manner and it seems important to carry out a less pragmatic study in order to answer general questions of the kind: what are the temporal constraint schemas which can be refined to equivalent regular transaction schemas? does a regular transaction schemas exists which does not corresponds to any temporal-logic-constraint schemas? These questions are obviously difficult because the parameters which have to be considered are complex. For instance, the choice to use SdetTL could be reconsidered, the limit to put on the use of auxiliary data structures has to be investigated as well as the restriction to regular languages of transactions.

References

[AB1] de Amo, S. / Bidoit N.: Contraintes Dynamiques d'Inclusion et Schémas Transactionnels. *In Neuviemes Journees Bases de Donnees Avancees*, 1993, pages 401-424.

[AB2] S. de Amo and N. Bidoit.: Operational Specification for Dynamic Functional Dependencies, Symposio Brasileiro de Banco de Dados, Recife (Bresil), pp 163-180, Oct. 1995.

[Ab] Abiteboul,S: Algebraic Analogues to Fundamental Notions of Query and Dependency Theory, Rapport de Recherche INRIA (1983).

[Am] S. de Amo.: Contraintes Dynamiques et Schemas transactionnels, PhD Thesis, University of Paris 13, Jan. 95.

[AHV] S. Abiteboul, R. Hull and V. Vianu.: Foundations of Databases. Addison-Wesley, 1995.

[AV1] Abiteboul, S / Vianu, V.: Transactions and Integrity Constraints. *In Proc. ACM SIGACT/SIGMOD Symp. on Principles of Database Systems*, 1985, pages 193-204.

[AV2] Abiteboul, S / Vianu, V.: Transactions Languages for Database update and specification. I.N.R.I.A. Technical Report 715, September 1987.

[AV3] Abiteboul, S / Vianu, V.: A Transaction-based Approach to Relational Database Specification. *Journal of the ACM*, 36, 4, October 1989, 758-789.

[AV4] Abiteboul, S / Vianu, V.: Procedural Languages for Database Queries and Updates. *In Proc. ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems*, Austin, 1988, pages 240-250.

[AV5] Abiteboul, S / Vianu, V.: The Connection of static constraints with determinism and boundedness of dynamic specification. *In Proc. of the Third Int. Conf. on Data and Knowledge Bases*, Jerusalem, 1988, pages 324-334.

[AWH] A. Aiken, J. Widom and J.M. Hellerstein.: Behavior of Database Production Rules: Termination, Confluence, and Observable Determinism. In Proceedings ACM-SIGMOD Intl. Conf. on Management of Data, pages 59-68, 1992.

[BK] A. Bonner and M. Kifer.: Transaction Logic Programming. Tech, report CSRI-270, April 1992 (revised June 1994), Computer Systems Research Institute, University of Toronto.

[BM] C. Beeri and T. Milo.: A Model for active object oriented databases. In Proceedings of Intl. Conf. on Very Large Data Bases, pages 337-349, 1991.

[BR] Brodie, M.L. / Ridjanovic, D.: On the design and specification of database transactions . *In M.L. Brodie, J. Mylopoulos and J.W. Schmidt, editors, On Conceptual Modelling*, pages 277-306. Springer-Verlag, 1984.

[C] Chomicki, J.: History-less Checking of Dynamic Integrity Constraints, *Int. Conf. on Data Engineering*, IEEE, Feb. 92, pp 557-564.

[CCF] Castillo, I.M.V. / Casanova, M.A. / Furtado, A.L.: A Temporal Framework for Database Specifications. *In Proc. Int. Conf. on Very Large Data Bases*, pages 280-291, 1982.

[CF] Casanova, M.A. / Furtado, A.L.: On the description of database transition constraints using temporal constraints. In *H. Gallaire, J. Minker and J.M. Nicolas, editors, Advances in Data Base Theory*, Vol.2, pages 221-236. Plenum Press, New York, 1984.

[CFP] Casanova, M.A / Fagin, R. / Papadimitriu, C.: Inclusion Dependencies and their Interaction with Functional Dependencies. In *Proc. 1st ACM SIGACT-SIGMOD Conf. on Principles of Database Systems* 1982, pp- 171-176.

[CFPT] S. Ceri, P. Fraternali, S. Paraboschi and L. Tanca.: Automatic Generation of Production Rules for Integrity Maintenance. In TODS 1994.

[CN] Chomicki, J. / Niwinski, D.: On the Feasibility of Checking Temporal Integrity Constraints, PODS 93.

[FP] P. Fraternali and S. Paraboschi.: *A review of Repairing Techniques for Integrity Maintenance.* In Proceedings of the workshop: Rules in Database Systems, Edinburgh, 1993. Springer-Verlag.

[FS] Fiadeiro, J. / Sernadas, A.: Specification and verification of Database Dynamics. In *Acta Informatica*, 25, pages 625-661, 1988.

[Li] Lipeck, U.W.: Transformation of Dynamic Integrity Constraints into Transactions Specifications. *ICDT'88 2nd International Conference on Database Theory*, 1988, LNCS 326.

[LR] Lassaigne, R. / de Rougemont, M.: Logique et Fondements de l'Informatique, Vol.1, Editions Hermès, Paris, 1993.

[LS] Lipeck, U.W. / Saake, G.: Monitoring Dynamic Integrity Constraints Based on Temporal Logic. *Information Systems* 12 (1987), pages 255-269.

[Ka] Kanellakis, P.C.: Elements of Relational Database Theory. *In Handbook of Theoretical Computer Science*, Vol. 2, pages 1073-1156.

[Ko] Koubarakis M.: Foundations of Temporal Constraint Databases, Doctoral Dissertation, Computer Science Division, Dept. of Electrical Engineering, National Technical University of Athens, Feb. 1994.

[Ma] Maier, D.: *The Theory of Relational Databases.* Computer Science Press (1983).

[Ni] Nicolas, J.-M.: Logic for improving integrity checking in relational databases. *Acta Inform.* 18, 227-253 (1982).

[Pn] Pnueli, A. Applications of Temporal Logic to the Specification and Verification of Reactive systems: A Survey of current Trends. L.N.C.S. 224, pp 510-584.

[PV] P. Picouet and V. Vianu.: Semantics and expressivness issues in active databases. In Proceedings of ACM Symposium on Principles of Database Systems, pages 126-138, San Jose, 1995.

[RU] Rescher, N. / Urquhart, A. Temporal Logic. Springer Verlag,1971.

[STSW] Schewe, K. D. / Thalheim, B. / Schmidt, J. W. and I. Wetzel: Integrity Enforcement in Object-Oriented Databases, Proc. 4th Int'l Workshop on Foundations of Models and Languages for Data and Objects, Oct. 1992.

[Su] Su, J.: Dynamic Constraints and Object Migration. *In Proc. of the 17th Int. Conf. on Very Large Data Bases*, September 1991, pages 233-242.

[Ull] Ullman, J.D.: Principles of Database Systems, 2nd Edition, Computer Science Press (1982).

[V] Vianu, V.: Dynamic Functional Dependencies and Database Aging. *Journal of ACM*, Vol. 34, 1, Jan 87, pp. 28-59.

[WC] J. Widom and S. Ceri.: Active Database Systems: Triggers and Rules for Advanced Database Processing. Morgan-Kaufmann, Inc., San Francisco, California, 1995.

[YP] Yannakakis, M. / Papadimitriou, C.H.: Algebraic Dependencies. *Journal of Computer and System Sciences* 25, 2-41, (1982).