Dynamic Integrity Constraint Evaluation in Temporal Deductive Databases

Maria Amélia Pacheco e Silva*
Universidade Federal de Uberlândia
Faculdade de Computação - João Naves Ávila, 2160
38408-100 - Uberlândia - MG - Brazil
+ 55 34 239 41 44
mamelia@ufu.br

Maria Ribera Sancho i Samsó
Universitat Politècnica de Catalunya
Departament de LSI - Jordi Girona Salgado 1-3
08034 - Barcelona - Catalonia - Spain
+ 34 93 401 71 11
ribera@lsi.upc.es

ABSTRACT

We propose a method for temporal integrity checking that consists in transforming a dynamic constraint to an equivalent past-evaluative form. The method proposed deals with integrity constraints defined by first-order logic formulas with explicit time in a denial form. It is based on the construction of time graphs to represent the temporal sequence of states presented in the constraint formula. We argue that our method can considerably facilitate the dynamic integrity constraint checking in relational or deductive temporal databases and can be considered as a step to optimize integrity checking in a temporal context.

Key words: Dynamic integrity constraints, specification and management of integrity constraints, past-directed formulas, temporal databases, deductive databases.

1. INTRODUCTION

Integrity constraints constitute a means for controlling the quality of the stored information therefore they play a very important role in the field of databases (DB). In the last twenty years many works related to integrity constraints (*constraints* or IC, for short) have been developed, most of them dealing with *static constraints* [1, 3, 4, 10, 12, 13, 18, 29], which are conditions that involve facts of only one state of the database, e.g., "A department's head must come from within the department".

Currently, in order to meet the needs of many applications, the area of *dynamic constraints* has also been investigated [2, 7, 9, 14, 25, 27, 28]. Dynamic constraints are conditions that involve facts of two or more states of the DB. They take into account the sequences of states that describe the dynamic behavior of the DB. An example is the condition "All new employees are assigned to some department within 30 days".

In this paper, we propose a method to simplify the dynamic integrity constraint checking in a temporal deductive context. This method can also be applied in a temporal traditional (relational) context. The dynamic integrity checking problem is considerably harder than static (non-temporal) case [6]. In the evaluation of a dynamic integrity constraint (DIC, for short) it should be assumed that the states considered in the constraint may range over any state of the DB, since we assume that the formula must be satisfied by the complete DB, and not only by each one of its states.

The naive solution is to make the DIC evaluation once we know all facts of the DB. If this were really the case, DICs would have little practical interest, since we should delay their evaluation until the end of system's lifetime, when all facts of the DB were known. A solution for this problem, as pointed by [20], is to reduce all DICs to a form that can be evaluated in each state, taking into account only the facts of the current and previous states. This form is called *past normal form*.

According to this reasoning, the DIC evaluation can be simplified if we reduce its formula to a past-directed form based on an instant T or, according to the convention of [20], a formula in a past normal form. A dynamic constraint in past normal form can be evaluated in a state T of the DB since the whole formula involves only facts that are available at T. Bearing in mind this goal, we have defined a method that transforms all dynamic constraint formulas to a past-directed form. This transformation will facilitate the dynamic constraint checking in temporal databases. The method presented in this paper is an extension of the work presented in [23] and [24].

The paper proceeds as follows. In Section 2, we introduce our temporal deductive data model and the language we use to specify dynamic constraints. In Section 3, we show how to construct time

*Currently working towards her Ph.D. degree in Computer Science at Technical University of Catalonia

ADVANCES IN DATA MANAGEMENT 2000
Krithi Ramamritham, T.M. Vijayaraman (Editors)
Tata McGraw-Hill Publishing Company Ltd.

graphs to represent the temporal sequence of the literals of the DIC formula. In Section 4, the method to transform a dynamic constraint into a past-evaluative formula is presented and illustrated with some examples. In Section 5, we compare our approach to some related work. Finally, in Section 6, we present conclusions and we discuss our perspectives for further work.

2. PRELIMINARIES

In this section we present some basic notions for our work. A convention we adopt in the paper is that the symbols [ ] represent an ordered sequence of elements and, as usual, the symbols { } represent sets, without any order of their elements.

2.1 Temporal Deductive Data Model

A temporal deductive database D consists of three finite sets: a set F of facts, a set R of deductive rules and a set I of temporal (static and dynamic) integrity constraints. A relational database is a deductive database without deductive rules. The set of facts is called the extensional database (EDB) and the set of deductive rules and temporal integrity constraints is called intensional database (IDB) [19].

Database predicates are either base or derived. A base predicate appears only in the EDB and eventually in the body of deductive rules. A derived predicate appears only in the IDB. Facts, rules and integrity constraints are formulated by a first-order logic language. We use names beginning with a lower case letter for predicate symbols and constants and a capital letter for variables.

We adopt a temporal model where the validity time of facts is explicit. Each information about the Universe of Discourse (UoD) is associated with a single time point, corresponding with the moment when the information holds in the UoD. This time point can be called the *occurrence time*.

In our work, time is represented by a discrete domain of time instants, which is isomorphic to the natural numbers. Each discrete time instant is simply denoted by a constant *t*. In our data model, t_0 denotes the starting time of the modeled application and t_f denotes the final time. The ordered sequence of consecutive time instants $T=[t_0,...,t_f]$ is called *system lifespan*. For all information *i* of the database, we suppose that $t(i) \in T$. Time points are expressed uniformly in a unique time unit (such as second, day, etc.) small enough to avoid ambiguities.

2.1.1 Facts in the Database

Facts appear in the EDB and are represented by base predicates. A fact is a ground atom, stored as positive instances $p(A_1,...,A_n,T)$, where the last term of the predicate *p* represents the occurrence time. The next example illustrates this type of predicates.

Example 2.1 - For a university database, we can store facts by base predicates like as:

offer(Course,Time)
enroll(Student,Course,Time)
transfer(Student,Course,Course,Time)

A fact *offer(c,t)* means that course *c* is offered as a new course at time *t*. A fact *enroll(s,c,t)* indicates that the student *s* enrolls in a course *c* at time *t*. A fact *transfer(s,cf,ct,t)* reports that student *s* is transferred from course *cf* to course *ct* at time *t*. □

2.1.2 Deductive Rules

A deductive rule is a formula of the form:

$$A \leftarrow L_1 \wedge ... \wedge L_n \quad \text{with } n \geq 1 \qquad (1)$$

where A is an atom, denoting the conclusion and each L_j is a literal representing conditions. Each L_j is either an atom or a negated atom. Any variables in A and L_j are assumed to be universally quantified over the whole formula. The terms in the conclusion are distinct variables and the terms in the conditions are variables or constants. Condition predicates may be ordinary or evaluable ("built-in"). The former are base or derived predicates. The latter are predicates such as the comparison or arithmetic predicates, which can be evaluated without accessing the database.

Example 2.2 - For the university case, we can define some derived predicates by means of the following deductive rules:

$$\text{takes}(S,C,T) \leftarrow \text{enrolled}(S,C,T_1) \wedge T_1 \leq T \wedge \neg(\text{transferred}(S,C,T_1,T)) \wedge \text{time}(T)$$
$$\text{enrolled}(S,C,T) \leftarrow \text{enroll}(S,C,T)$$
$$\text{enrolled}(S,C,T) \leftarrow \text{transfer}(S,Cf,Ct,T)$$
$$\text{transferred}(S,C,T_1,T) \leftarrow \text{enrolled}(S,C,T_1) \wedge \text{transfer}(S,C,Ct,T_2) \wedge T_2 > T_1 \wedge T_2 \leq T \wedge \text{time}(T)$$

These rules are self-explanatory. Note that predicate *enrolled* is defined by two deductive rules, since a student is enrolled in a course either by an enroll or by a transfer fact. □

2.1.3 Integrity Constraints

Integrity constraints are conditions that define properties to be satisfied by the DB in order to guarantee its integrity. Many languages for integrity constraint specification have been proposed. See [21] for a state-of-the-art survey.

Generally, integrity constraints are expressed by means of logic formulas. We represent an integrity constraint by a closed first-order logic formula in the form of a denial:

$$\leftarrow L_1 \wedge ... \wedge L_n \quad \text{with } n \geq 1 \qquad (2)$$

where the L_j are literals. More general constraints can be transformed into this form as described in [15].

For the sake of uniformity, we associate to each integrity constraint an inconsistency predicate *ic*, with or without terms, and thus it has the same form as the deductive rules. This representation for constraints has also been adopted in several methods of integrity constraint checking [11, 13, 16, 19, 29] and in some languages of conceptual modeling as Chimera [5], ROSES [8] and ODISSEA [26]. In this way, we rewrite the former denial as:

$$ic \leftarrow L_1 \wedge ... \wedge L_n \quad \text{with } n \geq 1 \qquad (3)$$

Because we adopt a data model where the validity time of facts is explicit, we are able to specify temporal constraints through first-order logic formulas with explicit time. As in [5, 16, 17, 25], we do not use temporal logic operators to define constraints. They are represented by formulas with temporal variables, which can refer to any state of the database: present, past or future. Temporal variables relate one to each other by means of comparison operators ($<, \leq, >, \geq, =$) or by arithmetic operators ($T_1 = T_2\text{-}1$, $T_1 = T_2+1$, etc.).

Example 2.3 - We can define integrity constraints for the university case by means of the following inconsistency predicates:

$ic_1(S,C,T,T_1) \leftarrow enroll(S,C,T) \wedge takes(S,C,T_1) \wedge T_1 < T$
$ic_2(S,Cf,Ct,T,T_1) \leftarrow transfer(S,Cf,Ct,T) \wedge takes(S,Ct,T_1) \wedge T_1 < T$
$ic_3(S,Cf,Ct,T,T_1) \leftarrow transfer(S,Cf,Ct,T) \wedge \neg(takes(S,Cf,T_1)) \wedge T_1 < T$

The inconsistency predicate *ic1* enforces the property that "a student *s* can be enrolled in a course *c* if he has not taken course *c* before". The inconsistency predicates *ic2* and *ic3* enforce the property that "a student *s* can be transferred from a course *cf* to a course *ct* if he has not taken course *ct* before and if he has taken course *cf* before". □

As the dynamic integrity constraint specification is a key concept in our work, we present it with more details in the next section.

2.2 Dynamic Constraint Specification

As commented before, we specify a dynamic integrity constraint through a closed first-order logic formula with explicit time, in a denial form. Therefore, we associate to each constraint ϕ of the DB an inconsistency predicate *ic*, defined by the integrity rule: $ic \leftarrow \neg \phi$. Thus, if ϕ is satisfied, the predicate *ic* will be false and if ϕ is violated, *ic* will be true. By means of a previous step of transformation of the general formula ϕ to a form of inconsistency predicate, we obtain two types of formulas for constraints.

The first type of constraints has only universal quantified variables in its formula. To simplify it, we eliminate the universal quantifiers and we suppose, as usual, that all free variables are universally quantified over the whole formula. Such constraint is called *universal constraint*.

However, in some cases, it is not possible to eliminate all the quantifiers of the formula. That occurs because, besides universally quantified variables, the inconsistency predicate has some existential quantifier. Universal quantifiers can be eliminated, as explained previously, but denied existential quantifiers stay in the formula. This constraint is called *existential constraint*. The formulas for the two kinds of constraints are presented next.

2.2.1 Dynamic Constraint in Universal Form

The inconsistency predicate of a dynamic constraint in universal form has the following formula:

$$ic(\overline{Y}, \overline{T}) \leftarrow r_1(\overline{Y}_1,T_1) \wedge ... \wedge r_j(\overline{Y}_j,T_j) \wedge \mathbf{[[}T_j \otimes T_{j-1}\mathbf{]]} \wedge ... \wedge r_m(\overline{Y}_m,T_m) \wedge \mathbf{[[}T_m \otimes T_{m-1}\mathbf{]]} \wedge \mathbf{[[}D(T_i,T_{i'}) \wedge C(\overline{Y})\mathbf{]]}$$
with $j = 1,...,m$; $m>1$ and $i,i' \in [1,...,m]$, with $i \neq i'$ (4)

where:

- $T_1,...T_j,...T_m$ are temporal variables that represent the time when a condition is true in the DB,
- $\overline{T}$ is a vector with all temporal variables of the formula, i.e., $\overline{T} = [T_1,...,T_j,...,T_m]$,
- $\overline{Y}$, $\overline{Y}_1$,... $\overline{Y}_j$,... $\overline{Y}_m$ are vectors of non-temporal variables, possibly empty, where $\overline{Y} = \overline{Y}_1 \cup ... \cup \overline{Y}_j \cup ... \cup \overline{Y}_m$,
- r_j is a literal or a conjunction of ordinary literals corresponding to the state represented by T_j,
- $\otimes$ is a comparison symbol among the temporal variables, belonging to the set $\{<, >, \leq, \geq\}$,
- $C(\overline{Y})$ is a literal or a conjunction of evaluable ("built-in") literals that represent comparisons among the non-temporal variables of the formula,
- $D(T_i,T_{i'})$ represents a literal or a conjunction of literals corresponding to evaluable predicates that establish a limited time interval between the states T_i and $T_{i'}$,
- All parameters among double brackets (**[[**.....**]]**) are optional.

The formula (4) uses a point-based representation of time; that is to say, the temporal variables of the formula refer to individual time instants. Therefore, this formula does not allow temporal variables relating to sets of time instants, which is an interval-based representation of time. Another feature of the formula is that each literal r_j has only one timestamp, represented by T_j, which indicates the time when a condition is true in the DB. Formulas with more than one timestamp per condition are not allowed.

Moreover, the formula (4) contains only conjunctions of literals because this form is more appropriate for our method. However, this feature does not restrict our work. If an integrity constraint has any operator of disjunction in its formula, we decompose the initial formula in several sub-formulas, in manner that each sub-formula has only conjunctions of conditions, as shown next:

$$ic_1 \leftarrow (L_1 \wedge ... \wedge L_j) \vee (L_{j+1} \wedge .... \wedge L_n)$$
$$ic_{1,1} \leftarrow L_1 \wedge ... \wedge L_j$$
$$ic_{1,2} \leftarrow L_{j+1} \wedge .... \wedge L_n$$

As can be noted, the formula (4) has temporal and non-temporal terms. Non-temporal terms of the formula are non-temporal variables, ordinary literals represented by r_j and evaluable literals represented by $C(\overline{Y})$. We assume that r_j can contain literals corresponding to basic or derived predicates, positive or negative. The non-temporal evaluable literals of C establish comparisons among the variables of $\overline{Y}$ by means of an arithmetic expression (such as $S_1 < S_2$) or by a predicate, for example: less_than (S_1,S_2), where S_1 and S_2 are non-temporal variables of the formula.

Temporal terms of the formula are temporal variables, evaluable literals represented by the expression $T_j \otimes T_{j-1}$ and evaluable literals represented by $D(T_i,T_{i'})$. Each expression $T_j \otimes T_{j-1}$ presents a comparison ($<, \leq, >, \geq$) among the temporal variables, determining a temporal sequence of the states represented in the formula. It is optional.

The term $D(T_i,T_{i'})$ is also optional but, if it appears in a formula, it establishes a limited time interval between the states represented by temporal variables T_i and $T_{i'}$. This term contains expressions such as $T_{i'} \varnothing T_i+d$ or $T_{i'} \varnothing T_i-d$, where $d \neq 0$ and $\varnothing \in \{=, <, \leq, >, \geq\}$. The term *d* of the previous expression, which is called *temporal distance*, is formalized in the definition 2.1.

Definition 2.1: Temporal Distance in a Dynamic Constraint Formula - A temporal distance *d* is a constant that appears in a temporal term of the DIC formula and expresses a value in absolute time units. A temporal distance establishes an interval of time between the states represented by $T_{i'}$ and T_i in a temporal term of the form: $T_{i'} \varnothing T_i+d$ or $T_{i'} \varnothing T_i-d$, where $\varnothing \in \{=, <, \leq, >, \geq\}$ and *d* is a positive integer number.

The specification of a temporal distance in the DIC formula can be illustrated by the temporal expression $T_2 > T_1 \wedge T_2 < T_1+10$. It means that the value of variable T_2 is greater than the value of variable T_1 and further T_2 is less than T_1 increased by 10 time units.

Example 2.4 - To illustrate the language we use in the specification of dynamic constraints in universal form, we present the formula for the condition "An employee's salary can never decrease". It can be specified in this language as:

$$ic(E,S_1,S_2,T_1,T_2) \leftarrow salary(E,S_1,T_1) \wedge salary(E,S_2,T_2) \wedge T_2 > T_1 \wedge S_2 < S_1$$

This formula represents a universal constraint. It contains two temporal variables: T_1 and T_2. It presents literals referring to two different states: salary(E,S_1,T_1) and salary(E,S_2,T_2). The relationship between the temporal variables is expressed by $T_2 > T_1$. This type of logic language provides great deal of freedom to express DIC. Temporal variables may range over any state. □

2.2.2 Dynamic Constraint in Existential Form

The inconsistency predicate of a dynamic constraint in existential form has the following formula:

$$ic(\overline{Y}'', \overline{T}'') \leftarrow \neg \exists \overline{Y}', \overline{T}' (r_1(\overline{Y}_1,T_1) \wedge ... \wedge r_j(\overline{Y}_j,T_j) \wedge [[T_j \otimes T_{j-1}]] \wedge ... \wedge r_m(\overline{Y}_m,T_m) \wedge [[T_m \otimes T_{m-1}]] \wedge [[D(T_i,T_{i'}) \wedge C(\overline{Y})]])$$

with $j = 1,...,m$; $m>1$ and $i,i' \in [1,...,m]$, with $i \neq i'$ (5)

where:

- $T_1,...T_j,...T_m$, $\overline{Y}_1,...\overline{Y}_j,...\overline{Y}_m$, rj, ⊗, $C(\overline{Y})$ and $D(T_i,T_{i'})$ are as described for formula (4),
- $\overline{T}'$ is a non-empty vector with all existentially quantified temporal variables, where $\overline{T}' \subseteq [T_1,...,T_j,...,T_m]$,
- $\overline{T}''$ is a vector, possibly empty, with all universally quantified temporal variables, i.e., $\overline{T}'' = [T_1,...,T_j,...,T_m] - \overline{T}'$,
- $\overline{Y}'$ is a vector, possibly empty, with all existentially quantified non-temporal variables, where $\overline{Y}' \subseteq (\overline{Y}_1 \cup ... \cup \overline{Y}_j \cup ... \cup \overline{Y}_m)$,
- $\overline{Y}''$ is a vector, possibly empty, with all universally quantified non-temporal variables, i.e., $\overline{Y}'' = (\overline{Y}_1 \cup ... \cup \overline{Y}_j \cup ... \cup \overline{Y}_m) - \overline{Y}'$,
- All parameters among double brackets (**[[**.....**]]**) are optional.

The formula of a dynamic constraint in existential form presents the same temporal and non-temporal terms as explained for the universal form in (4). The fundamental difference between the two kinds of formulas is that the formula in (5) contains an existential quantifier related with some of its temporal variables.

Also, there is another difference with respect to the arguments of the inconsistency predicate *ic*. These arguments indicate which values cause the constraint violation. Note that, in the universal form presented in (4), all temporal and non-temporal variables can be arguments of the inconsistency predicate, since all variables are quantified universally in the formula. However, the existential form in (5) presents as arguments of its inconsistency predicate only temporal and non-temporal variables that are not bound to any existential quantifier. The justification for that is explained in [20].

Respect to the existential quantifier at formula (5), note that it has a denied form (¬∃). We have considered this representation because it is more convenient for our method. By means of a procedure defined in [22] as a previous step for our method, we can transform any general DIC formula into one of the two forms of inconsistency predicate presented here: the universal form in (4), without any explicit quantifier, or the existential form in (5), with some denied existential quantifier.

Example 2.5 - To illustrate the language for specification of dynamic constraints in existential form, we present the constraint "All books borrowed by a student must be returned after at most 15 days". The formula of this constraint, considering a day as time unit, is the following:

$$ic(S,B,T_1) \leftarrow \neg \exists T_2 (borrows(S,B,T_1) \wedge returns(S,B,T_2) \wedge T_2 \geq T_1 \wedge T_2 \leq T_1+15)$$

This example illustrates the case of an existential constraint. It contains a temporal distance, represented by the integer 15. This temporal distance indicates a relative interval time of 15 days between the two states represented in the constraint formula. □

3. REPRESENTATION OF CONSTRAINTS THROUGH TIME GRAPHS

In this section, we introduce a key tool for our method that is useful in the temporal interpretation of DIC. We call it a *time graph of a formula*. A time graph represents the temporal sequence of the facts involved in the constraint and indicates the presence of existential quantifiers or temporal distances in the DIC formula. Its formal definition is presented next.

Definition 3.1: Time Graph of a Dynamic Constraint Formula - A time graph $\mathbf{T} = (G, L, R, D, Q)$ consists of:

- A directed graph G = (N, E), with a finite set of nodes (or vertices) N and a set of ordered pairs of edges $E \subseteq N \times N$.
- A node label L = (t, op) on N, with temporal variables t corresponding to the states represented in the DIC, and ordinary predicates op corresponding to facts of each state.

- An edge label R on E with comparison operators {<, ≤} to establish the type of relationship (strict or non-strict) between two vertices of each edge.
- An edge label D = (c, d) on E, with comparison operators c ∈ {=, <, ≤, >, ≥} and a temporal distance d. D is optional in a time graph.
- A node label Q on N with a denied existential quantifier. It is also optional.

A time graph has also a graphical representation, which will be presented shortly by several examples. The elements of a time graph sketch are nodes, arrows, predicate names and temporal variable names. The nodes represent the several states involved in the constraint. Each node is labeled in its inferior part by the temporal variable corresponding to each state. Its superior part is labeled by names of the corresponding predicates, with their respective non-temporal variables. The arrows represent the passage of time, from the occurrence of a fact until the occurrence of the next one. The states appear in an increasing order from left to right.

The time graph sketch can assume different aspects, depending on some features of the formula. These features are: if the formula has a universal or existential form, if there are temporal distances in the formula and, also, if the formula establishes or not a total order among its temporal variables.

3.1 Universal Time Graph

Time graphs can refer to constraints in universal or existential form. Depending on this condition, we call them *universal time graph* or *existential time graph*, respectively.

Definition 3.2: Universal Time Graph - A universal time graph is a graph $\mathbf{T} = ((N,E), L, R, D, Q)$ where Q is empty.

The simplest time graph of a constraint is constructed when the formula has a universal form and it is possible to establish a total order among all temporal variables. We call it a universal *linear time graph*. This graph assumes the appearance of an only straight line. This representation is possible because the facts represented in the formula have a linear sequence in the time. The formula contains explicit comparison predicates among all temporal variables or they can be deduced from the formula.

Definition 3.3: Universal Linear Time Graph - A universal linear time graph is a graph $\mathbf{T} = ((N,E), L, R, D, Q)$ where Q is empty, E and R are not empty and there is an instance in R for each E of the graph. Further, for all two ordered pairs in E: (Ni, Nj) and (Nk, Nm), Ni ≠ Nk and Nj ≠ Nm. However, Nj can be equal to Nk.

Example 3.1 - Let us consider a DIC with the following formula:

$$ic(X,T_1,T_2,T_3) \leftarrow r_1(X,T_1) \wedge r_2(X,T_2) \wedge r_4(X,T_2) \wedge T_2 > T_1 \wedge r_3(X,T_3) \wedge T_3 \geq T_2$$

Its time graph is presented in Figure 1.

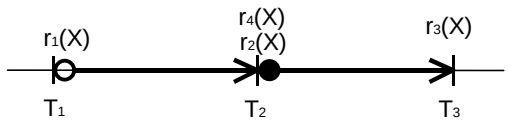

Figure 1. A universal linear time graph.

This time graph **T** has the following components:
- Directed graph G: G = ({1, 2, 3}, {(1, 2), (2, 3)})
- Node label L: L(1) = (T_1, (r_1(X))), L(2) = (T_2, (r_2(X), r_4(X))), L(3) = (T_3, (r_3(X)))
- Edge label R: R(1, 2) = <, R(2, 3) = ≤
- Edge label D: D = ( ) → empty
- Node label Q: Q = ( ) → empty
- It is a universal linear time graph.

By means of its graphical representation, we can see that predicate *r1* refers to the state represented by T_1, predicates *r2* and *r4* refer to the state represented by T_2 and predicate *r3* refers to the state represented by T_3. The sequence in the time is: the facts represented by predicates *r2* and *r4* happen after the fact represented by predicate *r1* and the fact represented by predicate *r3* happens after those represented by *r2* and *r4*.

With respect to the line between T_1 and T_2, it can be noted that the relationship is strict ($T_1 < T_2$). This strict relationship is represented in the graph by an empty circle in the state T_1 (o⟶), indicating that T_1 is less than T_2. The temporal relationship between T_2 and T_3 is non-strict ($T_2 \leq T_3$). It is represented in the graph by a depicted circle in T_2 (•⟶). So that, the time graph helps us to visualize the temporal sequence of the DIC clearly: $T_1 < T_2 \leq T_3$. □

Example 3.2 - The drawing of the time graph can be more complex. A factor that increases the complexity of the time graph is the presence of a temporal distance, such as in the following formula:

$$ic(X,T_1,T_2,T_3) \leftarrow r(X,T_1) \wedge s(X,T_2) \wedge T_2 > T_1 \wedge p(X,T_3) \wedge T_3 > T_2 \wedge T_3 \leq T_1+d$$

Note that this formula contains the evaluable literal $T_3 \leq T_1+d$ that specifies a temporal distance *d* between T_1 and T_3. The time graph for this formula is presented next.

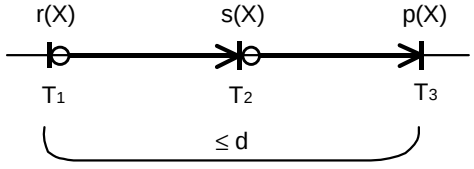

Figure 2. A linear time graph with a temporal distance.

By means of this time graph, we can see that the temporal sequence of the DIC is $T_1 < T_2 < T_3$ and that there is a time

interval, less than or equal to *d*, between the occurrence of the facts *r* and *p*. □

Now, we consider another kind of universal time graph, called *universal non-linear time graph*. This time graph also represents a formula without existential quantifiers. Another feature of this constraint is that the formula does not establish a total order among all temporal variables. There is not a relationship of comparison among all temporal variables of the formula. For this reason, the time graph cannot be represented by a single straight line, but for several lines that diverge starting from a point or that converge toward a final point.

Definition 3.4: Universal Non-Linear Time Graph - A universal non-linear time graph is a graph $\mathbf{T}$ = ((N,E), L, R, D, Q) where Q is empty, E and R are possibly empty. Further, if E are not empty, there are any two ordered pairs in E: (Ni, Nj) and (Nk, Nm) such that Ni = Nk or Nj = Nm.

Example 3.3 - An example of a constraint with a universal non-linear time graph can be:

$$ic(X,T_1,T_2,T_3,T_4) \leftarrow p(X,T_1) \wedge q(X,T_2) \wedge T_2 \geq T_1 \wedge r(X,T_3) \wedge T_2 > T_3 \wedge s(X,T_4) \wedge T_2 \geq T_4$$

This time graph $\mathbf{T}$ has the following components:
- Directed graph G: G = ({1, 2, 3, 4}, {(1, 2), (3, 2), (4, 2)})
- Node label L: L(1) = (T_3, (r(X))), L(2) = (T_2, (q(X))), L(3) = (T_1, (p(X))), L(4) = (T_4, (s(X)))
- Edge label R: R(1, 2) = <, R(3, 2) = ≤, R(4, 2) = ≤
- Edge label D: D = ( ) → empty
- Node label Q: Q = ( ) → empty
- It is a universal non-linear time graph.

Its graphical representation is presented next.

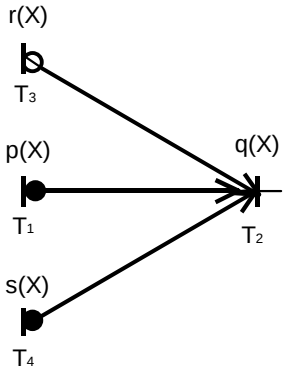

Figure 3. A non-linear time graph.

For this constraint, there are defined relationships between T_1 and T_2, between T_2 and T_3 and another between T_2 and T_4, but relationships between T_1 and T_3 are not specified, neither between T_1 and T_4, nor between T_3 and T_4. The time graph for this DIC shows clearly that the sequence of the facts represented by *r, s, p* and *q* are non-linear. The temporal sequence of this DIC is: $T_1 \leq T_2$, $T_3 < T_2$ and $T_4 \leq T_2$. □

The previous time graph is a *convergent non-linear time graph* because it is composed of several straight lines that converge toward an only point, i.e., the constraint defines an only final state (directed to the future). There is another type of non-linear graph, called *divergent non-linear time graph*. Such graph is composed by several straight lines that diverge from a point. Therefore, the graph does not contain a point that represents a single final state.

3.2 Existential Time Graph

All time graphs presented previously refer to constraint formulas without existential quantifiers. In this section we consider time graphs with existential quantifiers.

Definition 3.5: Existential Time Graph - An existential time graph is a graph $\mathbf{T}$ = ((N,E), L, R, D, Q) where Q is not empty.

All cases commented for a universal DIC can also be specified for a constraint in existential form, for example: DIC with temporal distance, without temporal distance, linear, non-linear, etc. For sake of space, we do not describe all these cases in the paper. We present only one case of existential time graph by means of an example.

Example 3.4 - This example illustrates the case of a DIC in existential form, with a divergent non-linear time graph and with a temporal distance. Its formula is presented below:

$$ic(X,T_1,T_2,T_4) \leftarrow \neg \exists\, T_3\, (r(X,T_1) \wedge s(X,T_2) \wedge T_2 > T_1 \wedge q(X,T_4) \wedge T_4 > T_2 \wedge p(X,T_3) \wedge T_3 > T_2 \wedge T_3 \leq T_2 + d)$$

This time graph $\mathbf{T}$ has the following components:
- Directed graph G: G = ({1, 2, 3, 4}, {(1, 2), (2, 3), (2, 4)})
- Node label L: L(1) = (T_1, (r(X))), L(2) = (T_2, (s(X))), L(3) = (T_4, (q(X))), L(4) = (T_3, (p(X)))
- Edge label R: R(1, 2) = <, R(2, 3) = <, R(2, 4) = <
- Edge label D: D(2, 4) = (≤, d)
- Node label Q: Q(4) = (¬∃)
- It is an existential non-linear time graph.

Figure 4 shows the outline of the graphical representation for this existential constraint.

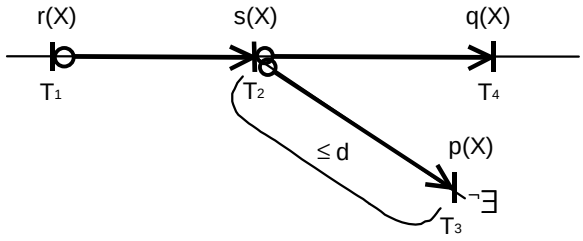

Figure 4. An existential time graph.

Note that it is necessary to put the quantifier "¬ ∃" in the time graph in order to represent the existential form of the constraint. This quantifier is relative to the state represented by the temporal variable T_3. □

4. METHOD FOR TRANSFORMATION OF DYNAMIC CONSTRAINTS INTO A PAST-EVALUATIVE FORM

In this section we define a method for DIC evaluation in temporal deductive databases, which is used to transform a dynamic constraint to an equivalent past-evaluative form. Constraints in past-evaluative form can be checked in each state of the DB, taking into account only the facts that are available in that state. In this way, our method can simplify the DIC checking in temporal databases.

In order to convert a dynamic constraint to a *past-evaluative (or past-directed) form*, it is necessary to take into account that a past or future notion is very relative and it depends on a reference point. We find an appropriate reference point for the DIC evaluation, such that, at this time, all facts involved in the formula are available. We label it the time T. Based on this time, the method rewrites the generic DIC formula. The new formula is a past-directed formula since it contains only facts from states [to,...,T].

4.1 Intuitive Notion

As we adopt a temporal representation of the DB states, the validity time of the facts is explicit. Each state is represented by a temporal variable in the DIC formula. Let us consider the following formula of a constraint:

$$ic(X,T_1,T_2) \leftarrow r(X,T_1) \wedge s(X,T_2) \wedge T_2 > T_1 \qquad (6)$$

The formula (6) involves two different states (represented by the variables T_1 and T_2), establishing a sequence between them. This sequence is defined by the expression $T_2 > T_1$. Based on this expression, we can construct the time graph to show the temporal sequence of the facts represented in the constraint, as shown in Figure 5.

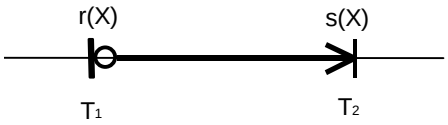

Figure 5. Graphical representation for a dynamic constraint.

Observing the graph, we can conclude that the constraint can be evaluated toward the past at time (or state) T_2, because, at this time, all facts represented in the DIC are available. We call it the *minimum time of evaluation* and we represent it in the graph (Figure 6) by the temporal variable T.

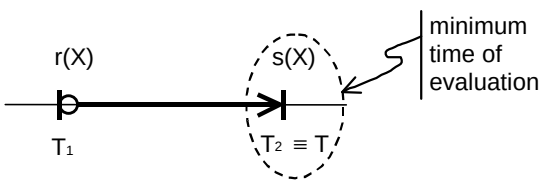

Figure 6. Minimum time of evaluation for a constraint.

After analyzing this time graph, we can rewrite the previous formula, reducing it to a past-directed formula. For that, we consider that the fact represented by *s* happens at current time T and that the fact represented by *r* has happened in a previous state (in the past). The new formula (rewritten from the instant T toward the past) is:

$$ic(X,T_1,T) \leftarrow s(X,T) \wedge r(X,T_1) \wedge T_1 < T \qquad (7)$$

Formulas (6) and (7) are equivalent since one is as a reflected mirror image of the other one. Therefore, the two formulas represent the same constraint, but with a different temporal interpretation. The formula (6) is a general formula. We cannot say if it is a future-direct or a past-directed formula because it depends on a reference time. On the other hand, the formula (7) represents a past-directed formula, with respect to the evaluation time T.

4.2 Minimum Time of Evaluation for a Constraint

In this section we present a definition for a key concept in our method. This key concept is the *minimum time of evaluation* for a constraint. This time is the reference point from which we can rewrite the DIC formula.

Definition 4.1: Minimum Time of Evaluation - Let T be the set of all temporal variables in the DIC formula. Let us consider that it is possible to establish a total order relation on T, symbolized by $P(T) = [T_1,.....,T_n]$. In this case, the relation P(T) defines an increasing sequence among the temporal variables of the formula: $T_1 \leq .... \leq T_i \leq .... \leq T_n$. The minimum time of evaluation of a constraint, called T, is defined as the last element of P(T): the element T_n, i.e., the greatest element in the ordered sequence.

When the order relation P(T) is not total, we obtain a set of "the greatest elements", i.e., several minimum instants of evaluation. For example, if P(T) is composed by the ordered sequences [T_1, T_2, T_3] and [T_1, T_2, T_4], the minimum time of evaluation T is represented by the set {T_3, T_4}.

However, when the time graph of the DIC is an existential graph, we cannot obtain the minimum time of evaluation directly for this process. There are other criteria to consider, for example, which temporal variable is bound to the denied existential quantifier and, if there is a temporal distance in the time graph, which position it occupies in this graph.

If there is a denied existential quantifier bound to one of the temporal variables that would represent the minimum time of evaluation (for the previous process), then we specify that the minimum time of evaluation is the final time of the system's life (t_f), i.e., $T = t_f$. This means that the DIC can be evaluated toward the past only at the end of the system's life. In an intuitive way, we can deduce that, if there is a quantifier $\neg\exists T_i$ in the DIC formula, and T_i is one of the minimum instants of the evaluation, then we need to retard its evaluation until the end of the system's life. This constraint will only be violated if some fact that must happen during the system's life, does not happen.

Another case is when the DIC time graph has the same characteristics mentioned up, but it also presents a temporal distance. Moreover, this distance restricts one of the temporal variables that represent one of the minimum instants of evaluation.

For example, let T be the set of the temporal variables of a DIC formula. Let us suppose that the order relation P(T) is the ordered sequence [T_1, T_2, T_3]. For the process mentioned up, T_3 is the minimum time of evaluation of the DIC. Let us also suppose that T_3 is existentially quantified ($\neg\exists T_3$) and that there is also a temporal distance between the variables T_1 and T_3, e.g., $T_3 \leq T_1+10$. In this case, the minimum time of evaluation T is equal to T_1+10.

In an intuitive way, one can see that $T = T_1+10$ must be the minimum time of evaluation because the constraint is sure to be violated (or not) only after an interval of 10 units of time after the instant T_1. At this time, all the necessary facts to the constraint checking are available.

4.3 Description of the Constraint Transformation Method

Previous sections have presented an intuitive notion of the method and a definition for the minimum time of evaluation of a DIC. Next, we describe our method, defining its steps.

Step 1) Building the time graph for the DIC formula, according to the instructions described in the Section 3.

Step 2) Determining the *minimum time of evaluation* for the DIC. This is the instant of minimum time in which one can evaluate the constraint toward the past, taking into account only the facts of this instant and facts of previous instants, as was defined in the Section 4.2.

Step 3) Building a new time graph for the constraint. This new graph contains the same ordinary literals of the initial graph, related to the facts that happen in the UoD. Literals are presented in this new graph in the same temporal order, if there was some order in the initial graph. Also, the *minimum time of evaluation*, defined in the previous step, is added to the graph. This minimum time of evaluation, a reference point for the formula conversion, is represented in the time graph by the temporal variable T. If there was some temporal variable T in the initial formula, we must substitute it by a new variable. For example, if the temporal variables of the formula are T_1, ..., T_j, ..., T_m, T, and T_j is the minimum time of evaluation, we substitute T by T_{m+1} and T_j by T, obtaining the temporal variables T_1, ..., T_{j-1}, T_{j+1}, ..., T_m, T_{m+1}, T.

Step 4) Rewriting a new formula for the DIC, based on the new time graph. This new formula is obtained by writing all ordinary literals of the time graph, putting first the literal that is related to the state represented by the temporal variable T and after, those referred to past states, in a right to left direction. The new formula contains all ordinary literals of the initial formula. However, it is a past-directed formula since it is written starting from the reference time T toward the past.

4.4 Analysis of Specific Cases for the Method

A key point of the method is the definition of the DIC minimum time of evaluation. According to the determination of this time, four cases of constraint evaluation appear: defined time evaluation, undefined time evaluation, end of life evaluation and end of interval evaluation.

4.4.1 Defined Time Evaluation Case

The first case is called *defined time evaluation*. This type of evaluation is possible when the constraint allows determining a total order among its temporal variables; that is to say, the minimum time of evaluation is explicitly defined by one and only one of the states represented in the formula. This state refers to the temporal variable of higher order in the sequence of time. As example we can present a constraint with the following formula:

$$ic(\ \bar{X}_1,\ \bar{X}_2,\ \bar{X}_3,T_1,T_2,T_3) \leftarrow r_1(\ \bar{X}_1,T_1) \wedge r_2(\ \bar{X}_2,T_2) \wedge T_2 > T_1 \wedge T_2 \leq T_1+d \wedge r_3(\ \bar{X}_3,T_3) \wedge T_3 > T_2$$

where $\bar{X}_1$, $\bar{X}_2$, $\bar{X}_3$ are vectors of non-temporal variables; T_1, T_2, T_3 are temporal variables; r_1, r_2, r_3 are literals or conjunctions of ordinary literals and *d* is a temporal distance among the states represented by T_1 and T_2. The time graph for the formula is presented below.

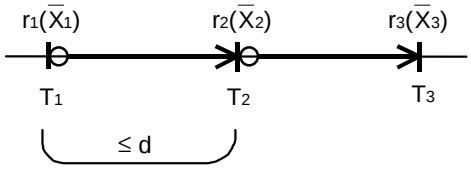

Figure 7. A constraint of the defined time evaluation case.

The minimum time of evaluation of this constraint is the state represented by the variable T_3 because there is a total order among the temporal variables P(T) = [T_1, T_2, T_3]. Therefore, at time T_3 we can evaluate the DIC, taking into account the fact of this state (r_3) and the facts of previous states (r_1 and r_2). The new time graph of this constraint is presented next.

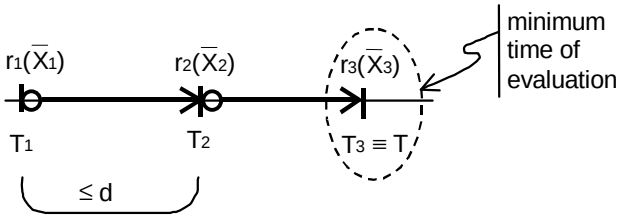

Figure 8. New graph for the defined time evaluation case.

Based on this time graph we rewrite a new formula for the DIC, which can be evaluated toward the past in the instant T. The new obtained formula is:

$$ic(\ \bar{X}_1,\ \bar{X}_2,\ \bar{X}_3,T_1,T_2,T) \leftarrow r_3(\ \bar{X}_3,T) \wedge r_2(\ \bar{X}_2,T_2) \wedge T_2 < T \wedge r_1(\ \bar{X}_1,T_1) \wedge T_1 < T_2\ \wedge T_1 \geq T_2\text{-}d$$

4.4.2 Undefined Time Evaluation Case

The second case is called *undefined time evaluation*. This evaluation can happen when the constraint does not allow determining a total order among its temporal variables; that is to say, the constraint presents a divergent non-linear time graph. As an example we can show a constraint with the following formula:

$$ic(\ \overline{X}_1,\ \overline{X}_2,\ \overline{X}_3,\ \overline{X}_4,T_1,T_2,T_3,T_4) \leftarrow r_1(\ \overline{X}_1,T_1) \wedge r_2(\ \overline{X}_2,T_2) \wedge T_2 > T_1 \wedge r_3(\ \overline{X}_3,T_3) \wedge T_3 > T_2 \wedge r_4(\ \overline{X}_4,T_4) \wedge T_4 > T_2$$

The time graph of this constraint is shown in Figure 9.

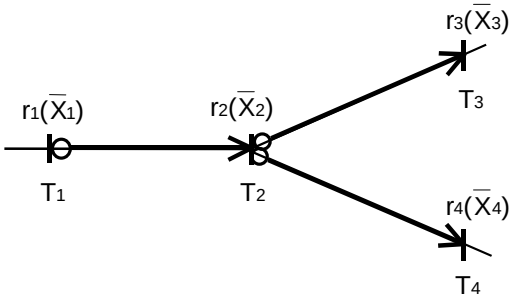

Figure 9. A constraint of the undefined time evaluation case.

This time graph presents two final states, represented by T_3 and T_4. It happens because the formula does not present a relationship among them. This DIC has a partial order among its temporal variables, represented by the ordered sequences [T_1, T_2, T_3] and [T_1, T_2, T_4]. The minimum time of evaluation is not unique; it is represented by the set {T_3, T_4}.

For this case, we define that the minimum time of evaluation is a fictitious variable T, such that $T \geq T_3$ and $T \geq T_4$. Thus, the constraint can be evaluated at T, taking into account the facts of the previous instants (r_1, r_2, r_3 and r_4). The new time graph is the following:

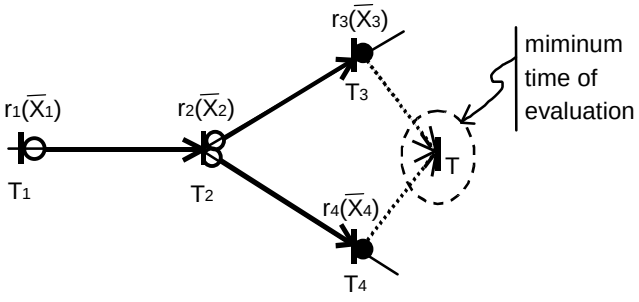

Figure 10. New graph for the undefined time evaluation case.

Based on this time graph we rewrite a new DIC formula, which can be evaluated toward the past in the instant T. The new obtained formula is:

$$ic(\ \overline{X}_1,\ \overline{X}_2,\ \overline{X}_3,\ \overline{X}_4,T_1,T_2,T_3,T_4,T) \leftarrow time(T) \wedge r_4(\ \overline{X}_4,T_4) \wedge T_4 \leq T \wedge r_3(\ \overline{X}_3,T_3) \wedge T_3 \leq T \wedge r_2(\ \overline{X}_2,T_2) \wedge T_2 < T_3 \wedge T_2 < T_4 \wedge r_1(\ \overline{X}_1,T_1) \wedge T_1 < T_2$$

4.4.3 End of Life Evaluation Case

The third case is called *end of life evaluation*. A DIC with this type of evaluation has a denied existential quantifier, bound to one of the temporal variables that would represent the minimum time of evaluation (for the process described previously). In this case, it is not possible to establish an instant T, before the end of the system's lifetime. This constraint is only violated if some fact that should happen during the system's lifetime, does not happen. Therefore, the only possible minimum time of evaluation is the final time t_f.

This constraint seems to have little practical value, since its evaluation should be retarded until the end of the system's lifetime. However we consider it because we intend to analyze all the possible cases in the transformation of DIC formulas. This type of DIC can be illustrated by means of the following example:

$$ic(\ \overline{X}_1,\ \overline{X}_2,\ \overline{X}_3,T_1,T_2) \leftarrow \neg\ \exists\ T_3\ (r_1(\ \overline{X}_1,T_1) \wedge r_2(\ \overline{X}_2,T_2) \wedge T_2 \geq T_1 \wedge r_3(\ \overline{X}_3,T_3) \wedge T_3 \geq T_2)$$

Its time graph is presented in Figure 11.

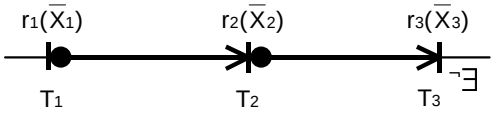

Figure 11. A constraint of the end of life evaluation case.

As the last state represented in the time graph is bound to a denied existential quantifier, this constraint will only be violated if there is a sequence of facts r_1 and r_2 in the DB and there is not a fact r_3 in any later state. This violation only can be confirmed at the end of the system's lifetime (t_f). The new time graph for the DIC is illustrated next.

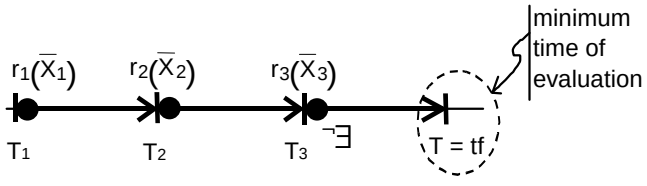

Figure 12. New graph for the end of life evaluation case.

Based on this time graph we rewrite a new formula for the DIC:

$$ic(\ \overline{X}_1,\ \overline{X}_2,\ \overline{X}_3,T_1,T_2,T) \leftarrow T=tf \wedge \neg\ \exists\ T_3\ (r_3(\ \overline{X}_3,T_3) \wedge T_3 \leq T \wedge r_2(\ \overline{X}_2,T_2) \wedge T_2 \leq T_3 \wedge r_1(\ \overline{X}_1,T_1) \wedge T_1 \leq T_2)$$

4.4.4 End of Interval Evaluation Case

The fourth case is called *end of interval evaluation*. A DIC with this type of evaluation has also a denied existential quantifier, bound to one of the temporal variables that would represent the minimum time of evaluation. Additionally, its formula presents some temporal distance that restricts one of the temporal variables, representing a minimum time of evaluation. As example, we show the following formula:

$$ic(\ \overline{X}_1,\ \overline{X}_2,\ \overline{X}_3,T_1,T_2) \leftarrow \neg\exists\, T_3\ (r_1(\ \overline{X}_1,T_1) \wedge r_2(\ \overline{X}_2,T_2) \wedge T_2 > T_1 \wedge r_3(\ \overline{X}_3,T_3) \wedge T_3 > T_2 \wedge T_3 \leq T_1+d)$$

The time graph of the constraint is presented next.

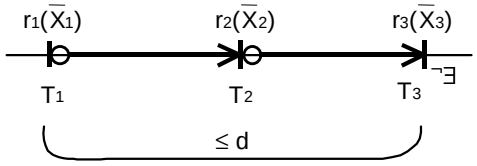

Figure 13. A constraint of the end of interval evaluation case.

In this case, the constraint can be evaluated, toward the past, only at the end of the time interval defined by the temporal distance *d* of the formula. Then, the minimum time of evaluation T is defined as an instant after *d* units of time after T_1, i.e., $T = T_1+d$. The new time graph is presented next.

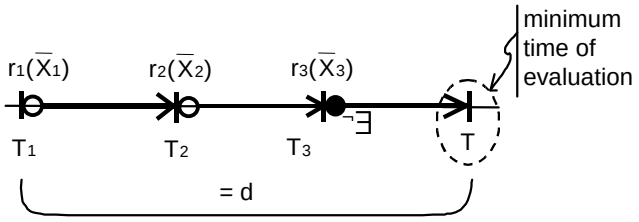

Figure 14. New graph for the end of interval evaluation case.

Based on this time graph, the new formula for the DIC is rewritten as:

$$ic(\ \overline{X}_1,\ \overline{X}_2,\ \overline{X}_3,T_1,T_2,T) \leftarrow time(T) \wedge \neg\exists\, T_3\ (r_1(\ \overline{X}_1,T_1) \wedge T_1 = T\text{-}d\ \wedge r_2(\ \overline{X}_2,T_2) \wedge T_2 > T_1 \wedge r_3(\ \overline{X}_3,T_3) \wedge T_3 > T_2 \wedge T_3 \leq T)$$

Besides the cases presented here, our method considers others types of dynamic constraints. The criteria we use to analyze constraints are: the number of states involved in the constraint, the presence of a denied existential quantifier bound to temporal variables, the existence of temporal distances in the formula and the type of the constraint time graph. During this analysis, 25 cases appear. However, the solution of these cases can be grouped in four types of evaluation, mentioned previously. Further details can be consulted in [23].

5. RELATED WORK

Despite the extensive research developed during the last decade [2, 5, 7, 9, 14, 16, 25, 27, 28], dynamic constraint enforcement has yet to become a practical technology. This is due to the lack of efficient methods for checking the satisfaction of dynamic constraints, since it is a very complex problem. Dynamic constraint checking requires expensive reasoning in multiples database states and has an inherent connection to the area of temporal logic and temporal reasoning. A survey of methods for definition and enforcement of dynamic constraints can be found in [21], where several methods are compared and classified with respect to some relevant features.

In this paper, we have proposed a method that can considerably facilitate the dynamic constraint checking in temporal databases. Many researches for solving this problem have been developed in the field of temporal databases. Most of these researches are conducted in the context of relational databases, e.g. [2, 7], object-oriented databases, e.g. [14, 27], or active databases, e.g. [9, 28]. Only a few works propose methods that consider temporal deductive databases [16, 25]. Our method is defined in a context of temporal deductive databases but it can also be applied in a relational database, since the latter is a deductive database with only facts and integrity constraints, without deductive rules.

Most of the existing methods for dynamic constraint checking express constraints by means of an implicit time temporal logic language [2, 7, 9, 14, 27, 28]. In this language, a dynamic constraint is represented through a formula of the first-order logic language extended with temporal operators (*since*, *until*, *previous* or ●, *next* or ○, *sometime in the past* or ◆, *sometime in the future* or ◇, etc.).

In the other hand, some methods use a logic language with explicit time to specify constraints [5, 16, 25]. In this case, each relation is augmented with a column holding the time instant of validity of each tuple. As we adopt a temporal model where the validity time of facts is explicit, we specify constraints through closed first-order logic formulas with explicit time. We do not use temporal operators in constraint formulas. All the references to time are explicit. We use variables and quantification over the time domain.

The two kinds of dynamic constraint specification have advantages and disadvantages. The explicit time approach can be a simpler and more natural way of representing constraints in temporal databases. However, the implicit time approach is a more elegant solution and its use is easier because we do not have to introduce any of the temporal variables; the references to time are encapsulated in the temporal operators [30].

On the other hand, the explicit time approach is more expressive than the implicit time approach. Some constraints expressible in the former are not expressible in the latter. As proved in [30], though an implicit time temporal logic language has more simplicity and ease of use, it cannot express all queries (and constraints) expressible using an explicit time logic language, in general.

Our work can handle dynamic constraints in universal or existential form, with or without temporal distances in the

formula. The work in [7] also consider temporal distances in the checking of real-time constraints but he uses an extension of Past FOTL (First-Order Temporal Logic), called Past Metric FOTL to formulate constraints.

A key tool in our method is the construction of time graphs as assistance in the temporal interpretation of the dynamic constraint. The works in [9], [14] and [27] also use a kind of graph for the monitoring of temporal integrity constraints, called *transition graph*. A time graph is similar to a transition graph because both are directed graphs, they can represent graphically a dynamic constraint and they can be used as a key for temporal constraint evaluation.

In spite of this, a time graph and a transition graph are different in several aspects. A time graph represents the temporal sequence of conditions present in the constraint. Nodes of this graph represent database states involved in the constraint. Each node is labeled in its inferior part by the name of the corresponding temporal variable. Its superior part is labeled by the names of the corresponding predicates, with their respective non-temporal variables. The arrows in the graph represent the passage of time, from the occurrence of a fact until the occurrence of the next one. A time graph can also indicate the presence of existential quantifiers and temporal distances in the formula.

On the other hand, transition graphs (an extended form of finite state machine) translate dynamic constraints on sequences of database states into conditions or single state transitions. Each node contains such parts of the constraint that remain to be monitored in the future of a present database state. The edges are labeled with static conditions that actually must be checked after each state transition. Differently from our work, in [9], [14] and [27] no explicit time attribute is associated with database states. Constraints are specified with Future PTL (Prepositional Temporal Logic) in [9] and [14], and with Past PTL in [27].

6. CONCLUSIONS AND FURTHER WORK

In this paper, we have defined a method to transform all dynamic integrity constraints to a past-evaluative form; that is to say, a formula that can be evaluated toward the past. We argue that our method can be considered as a step to simplify integrity checking in the context of temporal deductive databases.

Our method deals with integrity constraints defined by closed first-order logic formulas with explicit time in a denial form. We adopt formulas with a point-based representation of time; that is to say, the temporal variables of the formula refer to individual time instants. Moreover, formulas with more than one timestamp per condition are not allowed; each literal of the formula has only one timestamp.

We have also provided an approach to construct time graphs in order to represent the temporal sequence of states presented in the dynamic constraint formula. We have shown that time graphs can assume different aspects, depending on some formula features. For example, constraints can have a linear or non-linear time graph, depending on the existence of a total order among all temporal variables in the formula. To the best of our knowledge, the non-linear case has not been considered in any previous work in the field of integrity constraints.

The key concept of our method is the determination of a graph reference point in which the constraint can be checked by an evaluation directed toward the past, called the minimum time of evaluation of the constraint. According to this time, four cases of dynamic constraint evaluation appear, as explained and illustrated in the paper. Starting from this minimum time of evaluation, denoted by T, we rewrite a new formula for the constraint, which can be evaluated toward the past at time T.

In a future work, we intend to show an implementation of the method, at least for its more relevant cases. We are already working to prove that our method is correct and complete to make past-directed evaluation of any type of dynamic constraint. We are also making a classification of all possible kinds of constraints with respect to temporal reasoning in order to analyze if our method solves all existing cases.

7. ACKNOWLEDGEMENTS

Maria Amélia Pacheco e Silva was supported by a scholarship from CAPES (MEC/Brazil) during part of this work. The author is also very grateful to Antoni Olivé, Sandra de Amo and the anonymous reviewers for their useful comments and suggestions.

8. REFERENCES

[1] Bertino, E., Catania, B., and Bressan, S., "Integrity Constraint Checking in Chimera", Proc. 2nd Int. Workshop on Constraints Database Systems, Delphi, Greece, 160-186 (January, 1997).

[2] Bidoit, N., and De Amo, S., "Contraintes Dynamiques d'Inclusion et Schémas Transactionnels", Ingénierie des systèmes d'information, 2(1), 83-113 (1994).

[3] Bry, F., Manthey, R., and Martens, B., "Integrity Verification in Knowledge Bases", ECRC Report D.2.1.a, Munich, Germany, p. 26 (April, 1990).

[4] Ceri, S., and Widom, J., "Deriving Production Rules for Constraint Maintenance", Proc. 16th Int. Conf. On Very Large Data Bases, Brisbane, Australia, 566-577 (August, 1990).

[5] Ceri, S., Fraternali, P., and Paraboschi, S., "Constraint Management in Chimera", Data Engineering Bulletin, 17(2), 4-8 (1994).

[6] Chomicki, J., and Niwinski, D., "On the Feasibility of Checking Temporal Integrity Constraints", ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Washington, USA, 202-213 (1993).

[7] Chomicki, J., "Efficient Checking of Temporal Integrity Constraints Using Bounded History Encoding", ACM Transactions on Database Systems, 20(2), 149-186 (1995).

[8] Costal, D., Sancho, M. R., Olivé, A., and Roselló, A., "The Role of Structural Events in Behaviour Specification", Proc. 8th Int. Conf. on Database and Expert Systems Applications, Toulouse, France, 673-686

(September, 1997).

[9] Gertz, M., and Lipeck, U. W., "Deriving Optimized Integrity Monitoring Triggers from Dynamic Integrity Constraints", Data & Knowledge Engineering, 20(2), 163-193 (October, 1996).

[10] Grefen, P. W. P. J., and Apers, P. M. G., "Integrity Control in Relational Database Systems - An Overview", Data & Knowledge Engineering, 10(2), 187-223 (March, 1993).

[11] Griefahn, U., and Lüttringhaus, S., "Top-down Integrity Constraint Checking for Deductive Databases", Proc. 7th Int. Conf. On Logic Programming, Jerusalem, 130-144 (June 1990).

[12] Karadimce, A. P., and Urban, S. D., "A Framework for Declarative Updates and Constraint Maintenance in Object-Oriented Databases", Proc. 9th IEEE Int. Conf. on Data Engineering, Vienna, Austria, 391-398 (April, 1993).

[13] Kowalski, R., Sadri, F., and Soper, P., "Integrity Checking in Deductive Databases", Proc. 13th Int. Conf. on Very Large Data Bases, Brighton, England, 61-69 (September, 1987).

[14] Lipeck, U. W., and Saake, G., "Monitoring Dynamic Integrity Constraints Based on Temporal Logic", Information Systems, 12(3), 255-269 (1987).

[15] Lloyd, J. W., *Foundations of Logic Programming*, 2nd ed., Springer-Verlag, p. 212 (1987).

[16] Martín, C., and Sistac, J., "Applying Transition Rules to Bitemporal Deductive Databases for Integrity Constraint Checking", Proc. Int. Workshop on Logic in Databases, San Miniato, Italy, 117-134 (July, 1996).

[17] Mylopoulos, J., Borgida, A., Jarke, M., and Koubarakis, M., "Telos: Representing Knowledge about Information Systems", ACM Transactions on Information Systems, 8(4), 325-362 (October, 1990).

[18] Nicolas, J.-M., and Yazdanian, K., "Integrity Checking in Deductive Data Bases", In Gallaire,H.; Minker,J. (Eds.), *Logic and Data Bases*, Plenum Press, New York, 325-344 (1978).

[19] Olivé, A., "Integrity Constraints Checking in Deductive Databases", Proc. Of 17th Int. Conference on Very Large Data Bases, Barcelona, Catalonia, 513-523 (September, 1991).

[20] Olivé, A., "Integrity Constraints Specification", Technical report, LSI, Universitat Politècnica de Catalunya, Barcelona, Spain, p. 48 (May, 1995).

[21] Pacheco e Silva, M. A., "Dynamic Integrity Constraints Definition and Enforcement in Databases: A Classification Framework", Proc. of the First Working Conference on Integrity and Internal Control in Information Systems, Zürich, Switzerland, 65-87 (December, 1997).

[22] Pacheco e Silva, M. A., "Reducción de restricciones de integridad dinámicas a una forma evaluable hacia el pasado en modelos conceptuales temporales", PhD thesis, Departament de LSI, Universitat Politècnica de Catalunya, Barcelona, Spain. Forthcoming.

[23] Pacheco e Silva, M. A., and Sancho, M. R., "Transformación de restricciones de integridad dinámicas definidas hacia el futuro en una forma definida hacia el pasado", Technical report LSI-96-62-R, Universitat Politècnica de Catalunya, Barcelona, Spain, p. 54 (October, 1996).

[24] Pacheco e Silva, M. A., and Sancho, M. R., "Método de conversión de restricciones de integridad dinámicas en forma de futuro hacia una forma de pasado", Segundas Jornadas de Investigación y Docencia en Bases de Datos, Getafe, Spain, 183-184 (July, 1997).

[25] Plexousakis, D., "Compilation and Simplification of Temporal Integrity Constraints", Proc. 2nd Int. Workshop on Rules in Database Systems, Athens, Greece, 260-276 (September, 1995).

[26] Sancho, M. R., and Olivé, A., "The ODISSEA Approach to the Design of Information Systems from Deductive Conceptual Models. Extended version", Technical report LSI-93-16-R, Universitat Politècnica de Catalunya, Barcelona, p. 25 (1993).

[27] Schwiderski, S., Hartmann, T., and Saake, G., "Monitoring Temporal Preconditions in a Behaviour Oriented Object Model", Data & Knowledge Engineering, 14(2), 143-186 (December, 1994).

[28] Sistla, A. P., and Wolfson, O., "Temporal Conditions and Integrity Constraints in Active Database Systems", Proc. ACM-SIGMOD Int. Conf. on Management of Data, San Jose (California), USA, 269-280 (May, 1995).

[29] Teniente, E., and Olivé, A., "Updating Knowledge Bases while Maintaining their Consistency", The VLDB Journal, 4(2), 193-241 (1995).

[30] Toman, D., and Niwinski, D., "First-order Queries over Temporal Databases Inexpressible in Temporal Logic", Proc. 5th Int. Conf. on Extending Database Technology, Avignon, France, 307-324 (March, 1996).