SPECIFICATION, SEMANTICS, AND ENFORCEMENT OF DYNAMIC DATABASE CONSTRAINTS

H.-D.Ehrich, U.W.Lipeck, M.Gogolla

Institut für Informatik, Technische Universität Braunschweig
Postfach 3329, D-3300 Braunschweig, Fed.Rep.of Germany

ABSTRACT: In order to specify dynamic constraints, we present a simplified version of temporal logic based on the temporal quantifiers "always" and "sometime" as well as their bounded versions "always..until" and "sometime..before". We show that, in most practical cases, the bounded temporal quantifiers can be expressed by appropriate formulas with unbounded temporal quantifiers. We then use special kinds of temporal formulas as a language to specify dynamic constraints. The problem of enforcing such constraints is then reduced to the problem of enforcing dynamically changing sets of two kinds of static constraints, called universal and existential constraints. While universal constraints can be enforced strictly in principle, violation of existential constraints cannot be detected in each case at the earliest moment. We give a sufficient criterion for detecting violation of existential constraints.

1. INTRODUCTION

It is widely recognized that the specification and enforcement of constraints for databases is extremely important for the further development of database design and implementation. In information modelling, constraints are used to capture the peculiarities of real world situations and behaviour by giving appropriate rules. When running the database, these rules should be somehow obeyed in order to reduce the possibilities for incorrect data creeping in and corrupting the integrity of the data.

There are numerous papers addressing problems of integrity constraints for databases, and it is virtually impossible to give a comprehensive survey here appreciating each relevant contribution. For a treatment of constraints in several data models and data modelling approaches, the reader is referred to the recent textbook TL82 and its extensive bibliography. Also, ISO82 contains some material on comparing conceptual information modelling approaches. Among the earliest fundamental papers on constraints are HM75, St75,To77 and We76. Special sorts of constraints called dependencies play a dominant role in relational database theory. The reader is referred to Ma83 for a comprehensive treatment of this subject.

Recently, the problem of checking and monitoring integrity on the basis of constraints has found more and more interest (CD83,FDC81,Ni82,WSK83). Earlier approaches have been given in St75 and To77. CB80 considers the problem of verifying that transactions preserve constraints. Integrity checking in deductive databases is treated in NY78.

Most of the material on constraints published so far is concerned with static constraints, i.e. criteria to decide whether a given specific database state is admissible, disregarding its context between previous and forthcoming states. The area of dynamic constraints, i.e. criteria for admissible state sequences, is barely touched. Some preliminary ideas are contained in ISO82. Also, NY78 and Ri81 address this question, concentrating on pairs of before/after states.

A new approach to dynamic constraints using temporal logic has been initiated by CCF82,CF82 and GMS83. We take up these ideas here. However, by separating the problem of specifying and enforcing dynamic constraints from the problem of specifying and verifying transactions (which we do not consider in this paper), we restrict ourselves to a much simpler form of temporal logic incorporating only the temporal quantifiers "always" and "sometime" as well as their bounded versions "always..until" and "sometime..before". The latter, however, can be expressed by the unbounded versions in many cases of practical interest. Our approach is similar, but not identical, to that of Manna and Pnueli to program logic (MP81,Ma82).

Temporal logic is, in our opinion, a convenient tool for modelling dynamic database aspects. One of the problems widely discussed in this respect is the modelling of time in databases (An82, BADW82,Bu77,CW83). BADW82 gives a comprehensive overview of the role of time in information processing. In the temporal logic framework, time is modelled by considering state sequences. In particular, dynamic constraints determine admissible classes of state sequences in the same way as static constraints determine admissible classes of states. Static constraints are, of course, special cases of dynamic constraints.

Temporal logic must be based on a specific approach to concepts for formulating database schemata. We follow the lines of GMS83 and adopt a functional approach to data modelling in the spirit of BF79 and Sh81. Accordingly, a schema consists of sorts of entities, functions taking arguments and delivering results of specified sorts, and constraints expressed in temporal logic. It should be noted, however, that temporal logic is not bound to this approach to data modelling. It can be used with other approaches as well, e.g. relational ones as in CCF82 and CF82.

In contrast to previous functional approaches, we distinguish between two syntactic levels, the data level and the object level. The data level comprises specifications of basic data types like BOOL, INT, etc. The data level has a fixed interpretation that does not vary in time and is often the same for large classes of database schemata. The object level, on the other side, contains sorts like PERSON, PROJECT, etc. and functions on them whose interpretation varies in time, dependent on the database state. The object level will most probably be different for different database schemata.

In the next section, we give a brief account of our version of temporal logic, based on our functional approach to data modelling. We define the syntax and semantics of temporal formulas including the temporal quantifiers "always" and "sometime". It is well known (MP81) that these quantifiers enjoy the same nice duality principle as the classical quantifiers $\forall$ and $\exists$. We then introduce the bounded versions "always...until" and "sometime...before". They again enjoy a corresponding duality principle. Essentially, we use formulas to denote points in time. The idea is that a formula denotes that state in a given state sequence where it first becomes true. As formulas behaving especially well with respect to time, we define "monotonous" formulas that remain true if they once became true, We show that, with monotonous formulas as time bounds, the bounded versions of the temporal quantifiers can be expressed by appropriate unbounded formulas.

In the third section, we introduce special kinds of temporal formulas as a language to express dynamic constraints. We then show how the problem of enforcing such dynamic constraints can be reduced to the problem of enforcing dynamically changing sets of two kinds of static constraints, called universal and existential constraints, respectively. While universal constraints can be enforced strictly in principle (there are, however, considerable practical problems), existential constraints present principle problems. Our approach to enforcing existential constraints is on the safe side in so far as they are certainly violated if they are reported to be so, but violation cannot be detected in each case at the earliest moment.

2. TEMPORAL LOGIC

The syntax of a database schema is given by a collection of certain sorts and function names building the so-called schema signature. We subdivide it into a data part denoting basic data types like BOOL, INT, etc. and into an object part denoting the specific database in the style of the functional data model.

Formally, a schema signature $\Sigma = \langle S_D + S_O, \Omega_D + \Omega_O \rangle$ consists of:

- data sorts S_D including BOOL
- data functions Ω_D between data sorts
- object sorts S_O disjoint from S_D
- object functions Ω_O between object and data sorts, i.e. each $\omega \in \Omega_O$ has a (formal) arity $\omega : s_1 \times \ldots \times s_n \text{ --> } s_0$ with $s_i \in S_D + S_O$.

The data part $\langle S_D, \Omega_D \rangle$ may be fixed once, whereas the object part $\langle S_O, \Omega_O \rangle$ must be declared explicitly for any new schema.

Example: As an example, we consider a simple database for the registration and deregistration of cars. A complete verbal description of the schema can be found in ISO82; here, only self-explaining extracts are needed. The schema signature looks as follows:

- data sorts:	BOOL, INT, DATE, YEAR, ...
- data functions:	year-of: DATE --> YEAR, ...
- object sorts:	CAR, MANUF
- object functions:	

produced:	CAR --> BOOL
manufacturer:	CAR --> MANUF
serial-no:	CAR --> INT
registered:	CAR --> BOOL
date-of-reg:	CAR --> DATE
deregistered:	CAR --> BOOL
date-of-dereg:	CAR --> DATE
reg-no:	CAR --> INT
destroyed:	CAR --> BOOL
date-of-destr:	CAR --> DATE
today:	--> DATE

***

A schema signature may be interpreted by assigning certain sets to the sorts and appropriate functions on these sets to the function names; thus, a so-called schema instance is obtaines. The interpretation of the data part by basic data types is assumed to be fixed.

Let Σ be a schema signature. A $\underline{\Sigma\text{-instance}}$ or $\underline{\Sigma\text{-state}}$ $\sigma = \langle pos, act \rangle$ consists of two mappings:

- To each $s \in S_0$, a set pos(s) of "possible" values is assigned.
- To each $s \in S_0$, a set $act(s) \subseteq pos(s)$ of "actual" values choosen from the possible ones is assigned.
- Each function name $\omega: s_1 \times \dots \times s_n \text{ --> } s_0 \in \Omega_0$ is mapped to an "actual" function $act(\omega): act(s_1) \times \dots \times act(s_n) \text{ --> } act(s_0)$. (Especially, each function name with target BOOL denotes an actual relation.)

Such a state represents the contents of a database at a certain moment only, since the "actual" part may vary time-dependently. So the course of time in a database can be taken into account by observing sequences of states. Therefore, we will interpret a schema signature Σ by a $\underline{\Sigma\text{-state sequence}}$ $\underline{\sigma} = \langle \sigma_0 \sigma_1 \dots \rangle$ which denotes a (possibly infinite) sequence of Σ-instances $\sigma_i = \langle pos, act_i \rangle$ with the possible values implicitly given by a fixed mapping pos.

In order to restrict interpretations of a database schema to "admissible" states and state sequences only, some "constraints" are added to the schema signature Σ . E.g., in each state of our registration database, each car must be uniquely determined by its manufacturer and its serial-no. Also each car must be registered sometime after it has been produced; this condition must be reflected within a sequence of database states.

Such constraints will be expressed by temporal Σ-formulas defined as follows.

An $\underline{\text{atomic }\Sigma\text{-formula}}$ is a boolean Σ-term or an equation $t_1 = t_2$ between Σ-terms t_1, t_2 of the same sort. A $\underline{\text{nontemporal }\Sigma\text{-formula}}$ is constructed from atomic formulas by applying

- boolean connectives $\wedge$, $\vee$, =>, <=, $\neg$
- and quantifiers $\forall, \exists$ over individual variables.

A $\underline{\text{(temporal) }\Sigma\text{-formula}}$ is constructed from atomic formulas by applying (iteratively)

- boolean connectives and quantifiers as above
- quantifiers $\underline{\forall}$, $\underline{\exists}$ different from those above
- and unary "temporal operators" <u>always</u> and <u>sometime</u> .

Such a formula is called <u>closed</u> if each occurring variable is bound by some quantifier.

We assume the reader to know how to interpret a nontemporal Σ-formula φ in a Σ-state σ with a given substitution α of actual values for free variables; let $\sigma \models \varphi_\alpha$ denote that φ_α , i.e. the result of substitution, becomes true in σ . Of course, the quantifiers $\forall$, $\exists$ bind variables to set of actual values in a state (with sorts respected).

Temporal formulas, however, are interpreted in Σ-state sequences. Here, possible values may be substituted for variables, too; the different quantifiers denote the two kinds of binding. For a given Σ-state sequence $\underline{\sigma} = \langle \sigma_0 \sigma_1 \dots \rangle$ and a substitution α of possible values for free variables, the validity $\underline{\sigma} \models \varphi_\alpha$ is inductively determined by the rules (i)-(vi) below.

(i) An atomic formula φ only has to hold in the first state of the sequence $\underline{\sigma}$ provided that all values substituted are actual values in that state:

$\underline{\sigma} \models \varphi_\alpha$ iff:
all values occurring in φ_α exist in σ_0
and $\sigma_0 \models \varphi_\alpha$

(ii) Boolean connectives are interpreted as usual.

(iii) Quantifiers $\forall$, $\exists$ refer to all actual values in the first state of $\underline{\sigma}$.
E.g. take $\varphi \equiv \forall x\, \varphi'$:

$\underline{\sigma} \models \varphi_\alpha$ iff:
for all actual values v in σ_0:
$\underline{\sigma} \models \varphi'_{\alpha\langle x \leftarrow v \rangle}$

where $\alpha\langle x \leftarrow v \rangle$ substitutes x by v and agrees with α elsewhere.

(iv) In $\varphi \equiv \underline{\forall}\, x\, \varphi'$, however, all possible values are considered:

$\underline{\sigma} \models \varphi_\alpha$ iff:
for all possible values v: $\underline{\sigma} \models \varphi'_{\alpha\langle x \leftarrow v \rangle}$

($\underline{\exists}$ by analogy)

(v) For $\varphi \equiv$ <u>always</u> φ' , φ' must hold in any tail sequence of $\underline{\sigma}$ starting at an arbitrary state:

$\underline{\sigma} \models \varphi_\alpha$ iff:
for all i=0,1,... $\underline{\sigma}^i \models \varphi'_\alpha$

where $\underline{\sigma}^i = \langle \sigma_i\ \sigma_{i+1} \dots \rangle$.

(vi) For $\varphi \equiv$ <u>sometime</u> φ' , φ' must hold in at least one tail sequence of $\underline{\sigma}$:

$\underline{\sigma} \models \varphi_\alpha$ iff:
there exists i, $i \geq 0$, s.t. $\underline{\sigma}^i \models \varphi'_\alpha$

If the temporal operators are immediately applied to a nontemporal formula φ', rules (v) and (vi) say that φ' must hold in all states or in some state, respectively. By rule (i), an atomic formula φ_α yields false in a state if the formula involves objects not existing there. The validity of a compound nontemporal formula results according to rules (ii) and (iii).

For each schema signature Σ and each sort s, we assume a standard predicate "exists" which is defined by the formula:

$$\underline{\text{always}}\ \forall x\ \ \text{exists}(x)$$

Thus this predicate specifies which possible values v exist in the first state of a sequence $\underline{\sigma}$, since we get by rule (i):

$$\underline{\sigma} \models \text{exists}(v) \quad \text{iff} \quad v \text{ exists in } \sigma_0$$

We simply write "$\underline{\sigma} \models \varphi$", if a Σ-formula φ is valid in $\underline{\sigma}$ for all substitutions of possible values, and we write "$\models \varphi$", if φ is valid in all Σ-state sequences, assuming a fixed choice of possible values. These are purely semantical properties; we do not consider here syntactical deducibility of temporal formulas like it is known for nontemporal, i.e. first-order formulas (written "$\vdash \varphi$"). An axiomatization of a different kind of temporal logic has been presented in Ma82.

Obviously, the temporal operators are dual to each other under negation.

Prop.: Let φ be a Σ-formula.

$$\models \quad \neg\ \underline{\text{always}}\ \varphi \quad \Longleftrightarrow \quad \underline{\text{sometime}}\ \neg\ \varphi$$
$$\models \quad \neg\ \underline{\text{sometime}}\ \varphi \quad \Longleftrightarrow \quad \underline{\text{always}}\ \neg\ \varphi$$

For illustration, we list some formulas which ought to be valid for the car registration database as it develops dynamically. Let c, c_1, c_2 be variables of sort CAR, m of sort MANUF, and i of of sort INT.

(1) $\underline{\text{always}}\ \forall c_1\ \forall c_2$
$[\ \text{manuf}(c_1)=\text{manuf}(c_2)\ \wedge$
$\text{serial-no}(c_1)=\text{serial-no}(c_2) \Longrightarrow c_1=c_2\ ]$

(2) $\underline{\text{always}}\ \forall c_1\ \forall c_2$
$(\ \text{registered}(c_1) \wedge \text{registered}(c_2)\ )$
$\Longrightarrow\ (\ \text{reg-no}(c_1)=\text{reg-no}(c_2) \Longrightarrow c_1=c_2\ )$

(3) $\underline{\text{always}}\ \forall c$
$\text{produced}(c) \Longrightarrow \underline{\text{sometime}}\ \text{registered}(c)$

(4) $\underline{\text{always}}\ \forall c\ \forall m\quad [\ \text{manufacturer}(c)=m$
$\Longrightarrow \underline{\text{always}}\ (\text{exists}(c) \Longrightarrow \text{manuf.}(c)=m)\ ]$

(5) $\underline{\text{always}}\ \forall c\ \forall i\quad [\ \text{serial-no}(c)=i$
$\Longrightarrow \underline{\text{always}}\ (\text{exists}(c) \Longrightarrow \text{serial-no}(c)=i)\ ]$

Since an argument of a temporal operator normally refers to an unbounded state sequence, it is difficult to restrict that condition to a certain bounded part of the sequence. Typically a bound may be represented by the first occurrence of some other condition ψ, i.e. the first state in a sequence where ψ becomes true. Such situations have to be expressed in many applications.

Therefore, two additional binary temporal operators are introduced:

$$\underline{\text{always}}\ \varphi\ \underline{\text{until}}\ \psi$$
$$\underline{\text{sometime}}\ \varphi\ \underline{\text{before}}\ \psi$$

As arguments, temporal Σ-formulas φ and ψ are allowed. The semantics of the operators for a Σ-state sequence $\underline{\sigma}$ and a substitution α is given as follows, where

$$\mu_{\underline{\sigma}}(\psi_\alpha) \ = \ \min(\{\ j \mid \underline{\sigma}^j \models \varphi_\alpha\} \cup \{\infty\})$$

denotes the first occurrence of ψ.

(vii) $\underline{\sigma} \models (\underline{\text{always}}\ \varphi\ \underline{\text{until}}\ \psi)_\alpha$ iff:
for all i, $0 \leq i < \mu_{\underline{\sigma}}(\psi_\alpha)$: $\underline{\sigma}^i \models \varphi_\alpha$

(viii) $\underline{\sigma} \models (\underline{\text{sometime}}\ \varphi\ \underline{\text{before}}\ \psi)_\alpha$ iff:
there exists i, $0 \leq i < \mu_{\underline{\sigma}}(\psi_\alpha)$, s.t. $\underline{\sigma}^i \models \varphi_\alpha$

These definitions do not imply that the condition does ever become true. If wanted, this must be specified additionally by $\underline{\text{sometime}}\ \psi$. The new operators again behave dually.

Prop.:

$$\models \quad \neg\ \underline{\text{always}}\ \varphi\ \underline{\text{until}}\ \psi \Longleftrightarrow \underline{\text{sometime}}\ \neg\varphi\ \underline{\text{before}}\ \psi$$
$$\models \quad \neg\ \underline{\text{sometime}}\ \varphi\ \underline{\text{before}}\ \psi \Longleftrightarrow \underline{\text{always}}\ \neg\varphi\ \underline{\text{until}}\ \psi$$

Now we are able to state for our example, omitting the "$\underline{\text{always}}\ \underline{\forall}$"-prefix:

(6) $\text{reg-no}(c)=i$
$\Longrightarrow\ \underline{\text{always}}\ \text{reg-no}(c)=i\ \underline{\text{until}}\ \text{deregistered}(c)$

(7) $\text{destroyed}(c)$
$\Longrightarrow\ \underline{\text{always}}\ \neg\ \text{deregistered}(c)\ \underline{\text{until}}$
$\text{year-of}(\text{today}) \geq \text{year-of}(\text{date-of-destr}(c)) +3$

(8) $\text{destroyed}(c)$
$\Longrightarrow\ \underline{\text{sometime}}\ \text{deregistered}(c)\ \underline{\text{before}}$
$\text{year-of}(\text{today}) \geq \text{year-of}(\text{date-of-destr}(c)) +4$

Termination conditions (the "ψ" of the formulas above) in database specifications typically are given by arriving at some point of time; cf. (7), (8). Considering the irreversibility property of time we are especially interested in so-called monotonous conditions.

A nontemporal Σ-formula ψ is called <u>monotonous</u> w.r.t. a Σ-state sequence $\underline{\sigma}$ iff:

$$\underline{\sigma} \models \quad \psi \Longrightarrow \underline{\text{always}}\ \psi$$

Then the bounded temporal operators can be explained by the original temporal operators.

Prop.: Let ψ be monotonous wrt $\underline{\sigma}$.

(a) $\underline{\sigma} \models \underline{\text{always}}\ \varphi\ \underline{\text{until}}\ \psi \quad \Longleftrightarrow \quad \underline{\text{always}}\ \varphi \vee \psi$

(b) $\underline{\sigma} \models \underline{\text{sometime}}\ \varphi\ \underline{\text{before}}\ \psi$
$\Longleftrightarrow \quad \underline{\text{sometime}}\ \varphi \wedge \neg\ \psi$

Proof:

(a) Let α be an arbitrary substitution. The proposition is obvious in the case $\mu_{\underline{\sigma}}(\psi_\alpha) = \infty$; otherwise let $\mu := \mu_{\underline{\sigma}}(\psi_\alpha)$.

"==>": For all i, $0 \leq i < \mu$, we have $\underline{\sigma}^i \models \psi_\alpha$. Since $\underline{\sigma}^\mu \models \psi_\alpha$, montonicity gives for all $i \geq \mu$: $\underline{\sigma}^i \models \psi_\alpha$. Thus, for all $i \geq 0$: $\underline{\sigma}^i \models (\varphi \vee \psi)_\alpha$.

"<==": For all i, $0 \leq i < \mu$, we already know: $\underline{\sigma}^i \models (\varphi \vee \psi)_\alpha$. Since μ is the minimal j s.t. $\underline{\sigma}^j \models \psi_\alpha$, even $\underline{\sigma}^i \models \varphi_\alpha$ holds.

(b) By duality and (a), we conclude:

<u>sometime</u> φ <u>before</u> ψ

<==> $\neg$ <u>always</u> $\neg\varphi$ <u>until</u> ψ

<==> $\neg$ <u>always</u> $\neg\varphi \vee \psi$ <==> <u>sometime</u> $\varphi \wedge \neg\psi$

***

These transformations may be applied to the example formulas (6)-(8), if the following conditions are guaranteed: (d of sort DATE)

(9) deregistered(c) ==> <u>always</u> deregistered(c)

(10) today $\geq$ d ==> <u>always</u> today $\geq$ d

In the next section we will concentrate upon two special kinds of temporal formulas. Therefore, their semantics is given below.

<u>Remark:</u> Let φ and ψ be nontemporal formulas with free variables $x_1,\dots,x_n$. Then it can be concluded from the general rules:

(a) $\underline{\sigma} \models$ <u>always</u> $\underline{\forall}x_1\dots\underline{\forall}x_n$ (φ ==> <u>always</u> ψ)
iff for all $v_1,\dots,v_n$ (possible) and for all i=0,1,... with $\alpha(x_j):=v_j$ (j=1,...,n): $\sigma_i \models \varphi_\alpha$ implies that for all $k \geq i$: $\sigma_k \models \psi_\alpha$

(b) $\underline{\sigma} \models$ <u>always</u> $\underline{\forall}x_1\dots\underline{\forall}x_n$ (φ ==> <u>sometime</u> ψ)
iff for all $v_1,\dots,v_n$ (possible) and for all i=0,1,... with $\alpha(x_j):=v_j$ (j=1,...,n): $\sigma_i \models \varphi_\alpha$ implies that there exists k, $k \geq i$, s.t.: $\sigma_k \models \psi_\alpha$

3. DYNAMIC CONSTRAINTS

In this section, we introduce special kinds of temporal formulas as a language to express dynamic constraints, and we show how the problem of enforcing such dynamic constraints can be reduced to the problem of enforcing two dynamically changing sets of static constraints.

Let us first introduce a language which allows to express constraints. In this language a <u>database specification</u> $<\Sigma,C>$ consists of a signature and a set C of dynamic constraints. Each <u>dynamic constraint</u> is a temporal Σ-formula

φ ==> <u>always</u> ψ or

φ ==> <u>sometime</u> ψ .

where φ and ψ are nontemporal formulas.

A Σ-state sequence $\underline{\sigma}$ is <u>admissible</u> wrt C if each constraint in C is valid in all suffixes of $\underline{\sigma}$ for all substitutions of possible values. So the above constraints may be understood as abbreviations of formulas

<u>always</u> $\underline{\forall}x_1 \dots \underline{\forall}x_n$ (φ ==> <u>always</u> ψ) or

<u>always</u> $\underline{\forall}x_1 \dots \underline{\forall}x_n$ (φ ==> <u>sometime</u> ψ)

respectively, where $x_1,\dots,x_n$ are free in the bracketed parts. A constraint of the form

true ==> <u>always</u> ψ

can be considered as a <u>static constraint</u> ψ.

<u>Example:</u> We discuss some constraints for the car registration database that has partly been studied in the last section. The following rules shall be expressed by the constraints given below: After a car has been produced, it sometime must be registered before it is deregistered; once it is deregistered it cannot be registered again and a car cannot be registered and deregistered at the same time.

(1) produced(c) ==> <u>sometime</u> registered(c) <u>before</u> deregistered(c)

(2) registered(c) ==> <u>sometime</u> deregistered(c)

(3) deregistered(c) ==> <u>always</u> deregistered(c)

(4) produced(c) $\wedge$ registered(c) ==> $\neg$ deregistered(c)

(5) produced(c) $\wedge$ deregistered(c) ==> $\neg$ registered(c)

Due to the monotonicity in (3) the formula (1) is equivalent to :

(1') produced(c) ==> <u>sometime</u> [registered(c) $\wedge\neg$ deregistered(c)]

***

For databases, we feel that the above forms of dynamic constraints cover a wide range of applications. This is analogous to Hoare's program logic where formulas for pre- and postconditions of programs are restricted to $\{\varphi\}P\{\psi\}$ with P a program and φ, ψ predicates over P. If monotonicity of formulas is guaranteed by the remaining specification even constraints involving the operators "<u>always</u>...<u>until</u>" or "<u>sometime</u>...<u>before</u>" can be modified to formulas of the above form (as explained in section 2).

In order to reduce dynamic constraints to varying sets of static constraints, we introduce an implementation language for constraints. In this guage constructs of the form

<u>on</u> φ <u>do</u> op

will be used, where φ is a nontemporal formula (possibly with free variables) and op is an operation. The meaning of this is a kind of "trigger" activated when the value of the formula changes from false to true: if there is an substitution α to the free variables of φ in the present database state such that φ_α becomes true and φ_α was false in the previous state, then the operation op_α will be executed. op_α means that the operation uses the substitution α to bind the same free variables which occur in φ. In this sense the trigger "on φ do op" is parameterized wrt to all substitutions α such that φ_α becomes true. We call a collection of on-do constructs of the above form an <u>on-program</u> .

The operations op will manipulate two global variables C_u and C_e , which hold certain sets of nontemporal formulas where all free variables have been substituted by values of a database state. They represent the actual knowledge concerning the constraints.

- C_u is the set of <u>universal constraints</u> . A $\psi \in C_u$ has to be valid in all future database states.
- C_e is the set of <u>existential constraints</u> . It must be possible that each $\psi \in C_e$ can become true in some future database state.

C_u and C_e will change according to a given database state sequence $\langle\sigma_0\,\sigma_1\ldots\rangle$. C_u is a monotonous set in the sense that once a formula is in C_u , then it will always be. It may be, however, that a formula becomes redundant after inserting other formulas. For instance, $\psi_1 \vee \psi_2$ becomes obsolete after inserting ψ_1 since it then can be deduced. C_e is increasing and decreasing over time.

For a given set C of dynamic constraints the on-program induced by C is determined by the following rules, where "insert" and "delete" are the corresponding operations on sets:

(i) For each φ ==> <u>always</u> ψ in C take up:

<u>on</u> φ <u>do</u> insert(C_u,ψ)

(ii) Each constraint φ ==> <u>sometime</u> ψ induces:

<u>on</u> φ <u>do</u> <u>if</u> $\neg\psi$ <u>then</u> insert(C_e,ψ)
<u>on</u> φ <u>do</u> delete(C_e,ψ)

<u>Example:</u> Consider again the car database from above. We demonstrate how C_u and C_e are changing over time for a certain sequence of of states.

Let us start in a state where the car $\bar{c}$ has been produced but neither registered nor deregistered:

σ_0: produced($\bar{c}$)=true registered($\bar{c}$)=false deregistered($\bar{c}$)=false

C_u^0= { produced($\bar{c}$) $\wedge$ registered($\bar{c}$) ==> $\neg$deregistered($\bar{c}$) , produced($\bar{c}$) $\wedge$ deregisterd($\bar{c}$) ==> $\neg$registered($\bar{c}$) }

C_e^0= { registered($\bar{c}$) $\wedge$ $\neg$deregistered($\bar{c}$) }

- acc. constraint (1') -

Now assume that car $\bar{c}$ has been registered:

σ_1: produced($\bar{c}$)=true registered($\bar{c}$)=true deregistered($\bar{c}$)=false

C_u^1= C_u^0

C_e^1= { deregistered($\bar{c}$) }

- acc. constraint (2) -

After $\bar{c}$ has been deregistered, the following state is obtained :

σ_2: produced($\bar{c}$)=true registered($\bar{c}$)=true deregistered($\bar{c}$)=true

C_u^2= C_u^1 $\cup$ { deregistered($\bar{c}$) }

- acc. constraint (3) -

C_e^2= $\emptyset$

Now car $\bar{c}$ must always be deregistered, and that means that it cannot be registered again. (Additionally, constraint (3) demands that $\bar{c}$ must remain in the database) .

***

The admissibility of an actual database state will be checked in the following sense: If there is a universal constraint that is not valid in the actual state, then an exception condition is raised. Also, if an existential constraint contradicts to something deducible from the universal constraints, an error message is produced.

Of course, the latter test provides only sufficient conditions but not necessary conditions for the existence of an admissible continuation for the present sequence of states. The condition for the validity of the existential constraints C_e can be expressed as:

$\forall\psi \in C_e\ \exists\sigma_\psi$ s.t. $\sigma_\psi \models C_u^\psi$ and $\sigma_\psi \models \psi$ and σ_ψ is reachable from the actual state

Here, C_u^ψ denotes the set of universal constraints in the state σ_ψ. If $\neg\psi$ is deducible from (the actual) C_u , this will hold in all future states due to the monotonicity of C_u , i.e.: $\sigma_\psi \models C_u^\psi$ and $\sigma_\psi \models \psi$ cannot become true in a future state σ_ψ , whether or not σ_ψ can be reached.

To be a little more precise, C_u and C_e are characterized below for a given admissible state sequence $\langle \sigma_0 \ldots \sigma_n \rangle$. After C_u and C_e have been updated by the on-program according to the rules (i)-(ii) and an admissibility check in the above sense has been performed successfully, the following invariant conditions hold for C_u and C_e:

$\psi_\alpha \in C_u$ iff: $\exists$ ($\varphi \Longrightarrow$ <u>always</u> ψ) $\in C$
  $\exists$ substitution α for the free variables in φ, ψ
  $\exists$ $i \leq n$ s.t. $\sigma_i \models \varphi_\alpha$ and
    for all k, $i \leq k \leq n$: $\sigma_k \models \psi_\alpha$

$\psi_\alpha \in C_e$ iff: $\exists$ ($\varphi \Longrightarrow$ <u>sometime</u> ψ) $\in C$
  $\exists$ substitution α for the free variables in φ, ψ
  $\exists$ $i \leq n$ s.t. $\sigma_i \models \varphi_\alpha$ and
    for all k, $i \leq k \leq n$: [not($\sigma_k \models \psi_\alpha$)]
    and [($C_u \vdash \neg\psi_\alpha$) does not hold]

This means that the dynamic "<u>always</u>" constraints are enforced strictly by enforcing the universal constraints in C_u . The dynamic "<u>sometime</u>" constraints are enforced as closely as is perhaps practically feasible: an existential constraint $\psi_\alpha \in C_e$ is considered violated if it contradicts the current universal constraints and cannot become true any more for this reason. This does not mean, however, that ψ_α really can become true in a future state if it is not considered violated. It might be that the remaining specification already excludes any admissible continuation in principle or that the next state transitions do so data-dependently.

4. CONCLUSIONS

The simplified temporal logic that we propose here for specifying dynamic constraints has the advantage of being "implementable" in the sense that dynamic constraints specified this way can be enforced along the lines indicated above. There are, of course, considerable practical difficulties in implementing integrity monitors with feasible efficiency. Enforcing what we called universal constraints has been investigated by some authors, e.g. CB80,CD83,FDC81,Ni82, St75,To77, and WSK83. But still, practical solutions are either very restricted or very inefficient. The problem of enforcing what we called existential constraints still is much harder. It seems to have been ignored up to now. Its solution requires a great amount of deduction in a logical system. So this problem contributes to the need for powerful deductive data bases. Perhaps, direct hardware support can help to achieve feasible solutions for this problem.

On the theoretical side, it would be nice to have a strict criterion for violation of an existential constraint at the earliest possible moment. The problem is to decide on the basis of a current state whether a formula can or cannot become true in some reachable future state. Such a criterion would have to take the set C of dynamic constraints itself into account, since these can give rise to new universal constraints in subsequent states, possibly causing inconsistencies with the present existential constraints. Moreover, the problem of reachability of states has to be taken into account, and this depends on the update operations available.

REFERENCES

(An82) Anderson,T.L.: Modelling Time at the Conceptual Level. In: Improving Database Usability and Responsiveness, Proc. 2nd Int. Conf. on Databases (P.Scheuermann,ed.), Academic Press, New York 1982, 273-297

(BADW82) Bolour,A./Anderson,L./Dekeyser,L./ Wong,H.: The Role of Time in Information Processing. SIGMOD Record 12,3 (1982), 27-50

(BF79) Buneman,P./Frankel,R.E.: FQL - A Functional Query Language. Proc. ACM SIGMOD Int.Conf. on Management of Data 1979, 52-58

(Bu77) Bubenko,J.: The Temporal Dimension in Information Modelling. In: Architecture and Models in Data Base Management Systems (G.Nijssen, ed.), North-Holland, Amsterdam 1977, 93-118

(CB80) Casanova,M.A./Bernstein,P.A.: A Formal System for Reasoning about Programs Accessing a Relational Database. ACM TOPLAS 2 (1980),386-414

(CCF82) Castilho,J.M.V./Casanova,M.A./Furtado, A.L.: A Temporal Framework for Database Specifications. In: Proc. 8th Int. Conf. on Very Large Data Bases, Mexico 1982

(CD83) Cremers,A.B./Domann,G.: AIM, An Integrity Monitor for the Database System INGRES. Proc. 9th Int.Conf. on Very Large Data Bases (M.Schkolnick/C.Thanos,eds.), Florence 1983, 167-170

(CF82) Casanova,M.A./Furtado,A.L.: A Family of Temporal Languages for the Description of Transition Constraints. In: Proc. Workshop on Logical Bases for Data Bases, Toulouse 1982

(CW83) Clifford,J./Warren,D.S.: Formal Semantics for Time in Data Bases. ACM TODS 8 (1983), 214-254

(FDC81) Furtado,A.L./dosSantos,C.S./Castilho, J.M.V.: Dynamic Modelling of a Simple Existence Constraint. Inform. Syst.6 (1981), 73-80

(GMS83) Golshani,F./Maibaum,T.S.E./Sadler,M.R.: A Modal System of Algebras for Database Specification and Query/Update Language Support. Proc. 9th Int. Conf. on Very Large Data Bases, Florence 1983, 331-339

(HM75) Hammer,M.M./McLeod,D.J.: Semantic Integrity in a Relational Database System. Proc. Int. Conf. on Very Large Data Bases, 1975, 25-47

(ISO82) ISO/TC97/SC5/WG3: Concepts and Terminology for the Conceptual Schema and the Information Base (J.J.van Griethuysen,ed.), ISO/TC97/SC5/WG3-N695, 1982

(Ma83) Maier,D.: The Theory of Relational Databases. Pitman, London 1983

(Ma82) Manna,Z.: Verification of Sequential Programs: Temporal Axiomatization. In: Theoretical Foundations of Programming Methodology (M.Broy/G.Schmidt,eds.), Reidel Publ.Co., Dordrecht 1982, 53-101

(MP81) Manna,Z./Pnueli,A.: Verification of Concurrent Programs: The Temporal Framework. In: The Correctness Problem in Computer Science (R.S. Boyer/J.S.Moore,eds.) Academic Press,London 1981, 215-273

(Ni82) Nicolas,J.M.: Logic for Improving Integrity Checking in Relational Databases. Acta Informatica 18 (1982), 227-253

(NY78) Nicolas,J.M./Yazdanian,K.: Integrity Checking in Deductive Databases. In: Logic and Databases (H.Gallaire/J.M.Nicolas,eds.), Plenum Press, New York 1978, 325-344

(Ri81) Richter,G.: Utilization of Data Access and Manipulation in Conceptual Schema Definitions. Inform. Syst. 6 (1981), 53-71

(RU71) Rescher,N./Urquhart,A.: Temporal Logic. Springer, Berlin 1971

(Sh81) Shipman,D.W.: The Functional Data Model and the Data Language DAPLEX. ACM TODS 6 (1981), 140-173

(St75) Stonebraker,M.: Implementation of Integrity Constraints and Views by Query Modification. Proc. ACM SIGMOD Int. Conf. on Management of Data, San Jose 1975, 65-78

(TL82) Tsichritzis,D.C./Lochovsky,F.H.: Data Models. Prentice Hall, Englewood Cliffs 1982

(To77) Todd,S.:Automatic Constraint Maintenance and Updating Defined Relations. Proc. IFIP Congress 77(B.Gilchrist,ed.), North-Holland, Amsterdam 1977, 145-148

(We76) Weber,H.: A Semantic Model of Integrity Constraints in a Relational Database. In: Modelling in Database Management Systems (G.Nijssen, ed.), North-Holland, Amsterdam 1976, 269-292

(WSK83) Weber,W./Stucky,W./Karszt,J.: Integrity Checking in Database Systems. Inform. Syst. 8 (1983), 125-136