Using Finite–Linear Temporal Logic for Specifying Database Dynamics

Gunter Saake
IBM Wissenschaftliches Zentrum Heidelberg, Tiergartenstr. 15, 6900 Heidelberg
Udo W. Lipeck
FB Informatik (VI), Universität Dortmund, Postfach 500500, 4600 Dortmund 50

1 Introduction

The specification of a database system consists of the description of its static information structure as well as of its dynamic behaviour. Whereas in classic conceptual database design the main interest was on the static part, specification of database dynamics became an important topic in the last few years.

The specification of dynamic database behaviour has to describe the correct evolutions of the stored information. This can be done in terms of allowed user actions, which update database states, or in a descriptive manner by specifying the correct sequences of database states. Here, we concentrate on the latter aspect and present a corresponding logical specification calculus.

Starting with [Se80], the use of a *temporal logic framework* for specifying database dynamics was proposed by several working groups, among them [CF84, Ku84, SFNC84, ELG84, LEG85, KMS85, CS87]. In these approaches, the semantical interpretation of database states by first order logic models is extended in a natural way to the interpretation of database dynamics by sequences of database states. This enables the use of classical first order logic for specifying static databases structure together with a temporal logic description of the desired database behaviour in a single specification framework.

Similar to [SFNC84, CS87], we use a *layered approach* for database specification. A conceptual database schema is described by the following four specification levels :

1. The *Data Level* describes the basic data structures, for examples the domains `integer` or `point`, in terms of abstract data types. These are usually given by means of algebraic (equational) specifications.
2. On the *Object Level* the information stored in the database is described. Typical examples are object classes like `PERSON` or `CITY`. The object level corresponds to the conceptual data model in popular database design methodologies.

   In contrast to the data items of the data level, the object population as well as the object properties are varying in time.
3. The *Behaviour Level* describes the allowed databases evolutions in a descriptive fashion. Temporal logic is used as specification language.

4. As last level, the *Action Level* describes the basic database manipulations to be used in applications. These actions are often specified using pre- and postconditions.

A multi-levelled conceptual database schema reflects two complementary views of database dynamics. The first view globally describes the correct database evolutions by *temporal integrity constraints.* An example is the requirement "Salaries may never decrease". Temporal integrity constraints are usually long-term restrictions on allowed object modifications. The second view locally specifies the allowed database state changes by *action specifications.* A typical example is "Increase salary of employee 'Smith' by x percent".

Both views together prescribe *correct database behaviour* : Correct database evolutions have to satisfy the temporal integrity constraints and must be induced by a sequence of actions. For the examples above, in a correct database evolution the salary of an employee cannot be modified by a negative value for x.

As mentioned earlier, we concentrate on the behaviour level. Using the temporal logic formalism, the example constraint "The salary of employees may never decrease." looks as follows :

```
FOR ALL x : EMPLOYEE
  FOR ALL y : integer
    ALWAYS ((sal(x)=y) ⇒ ALWAYS(sal(x)≥y))
```

In contrast to other applications of temporal logics, we have specific requirements for their use in database theory. Database states correspond to models of a many–sorted first order logic. This enables the use of logic oriented data retrieval languages as well as constraints formulated in usual first order logic. The logic oriented approach is necessary also if one wants to extend databases by deductive capabilities. It follows that we need a first order temporal logic.

Semantic models for *database dynamics* must be able to express the usual database operations

- insertion of database objects,
- deletion of database objects and
- update of database object properties.

We choose the obvious approach, that *different* database states correspond to *different* models having *changing* object sort domains. Other first order temporal logics [MP81, Krö87] use a constant model at all points of time and are not suited for modelling database semantics. This makes it necessary to define a new first order temporal logic to handle this specific requirements. In this paper we concentrate on the usage of this logic in database specification, a more detailed overview on first order temporal logics with changing domains can be found in [Sa88b].

The rest of the paper is organized as follows. The next chapter formally defines the interpretation structures used in our specification framework. It is followed by the definition of syntax and semantics for the temporal logic used in behaviour specifications. There a few useful properties of this logic are explained, too. Finally, by means of transition graphs basic techniques for the analysis and implementation of temporal logic specifications are discussed.

This paper is based on a detailed elaboration in the recent thesis [Sa88a].

2 Semantic Models

The formal semantics of a database state is a model of a many–sorted first order logic. This allows the use of logic based retrieval and constraint languages: Static database constraints are expressed as first order formulae restricting the possible database states. The sorts and the non–logical symbols of the many–sorted first order logic are determined by the database schema. A *signature* $\Sigma = (S, \Omega, \Pi)$ consists of

- a set of sorts $S = \{s, r, \ldots\}$,
- a set of functions $\Omega = \{f\colon s_1 \times \ldots \times s_n \rightarrow s_0,\ \ g\colon r_1 \times \ldots \times r_m \rightarrow r_0,\ \ldots\}$
- and a set of predicates $\Pi = \{p\colon s_1 \times \ldots \times s_n,\ \ q\colon r_1 \times \ldots \times r_m,\ \ldots\}$ containing a binary equality predicate $=_s\colon s \times s$ for each $s \epsilon S$.

An *interpretation structure* A_Σ for Σ maps each sort $s \epsilon S$ to a set $A_\Sigma(s)$ and interprets the function and predicate symbols by functions $A_\Sigma(f)$ and predicates $A_\Sigma(p)$ over the sort carriers $A_\Sigma(s)$. If unambiguity is provided, we simply write A instead of A_Σ. Further, we use interpretation structures as mathematical models of database states and use these notations alternatively. Together with a set $X = \{x\colon s,\ y\colon r, \ldots\}$ of typed variables, a signature determines the formulae of a predicate logic.

A variable x of sort s takes elements from the sort carrier $A_\Sigma(s)$ as possible values. For a given database state, a *substitution* ϑ is a mapping from the variables in X into the related sort carriers. The value of terms and atomic formulae for a given interpretation structure and for a given substitution is uniquely determined by the above definitions and the well known term evaluation methods. As usual, syntax and semantics of first order predicate logic formulae can be defined using these conventions.

The semantic extension to state sequences is achieved by using *Kripke structures.* A Kripke structure consists of a set W of *worlds*, a mapping of these worlds into database states and a *reachability relation* R between the worlds. Additionally, one of the worlds is marked as the *initial* one. In formal terms, a Kripke structure K_Σ wrt a given signature Σ is defined by

- a set $W = \{\alpha, \beta, \ldots\}$ of *worlds*,
- a *reachability relation* $R \subseteq W \times W$,
- an *interpretation function* I mapping each world α to an interpretation structure $I(\alpha)$ for the signature Σ and
- an *initial world* α_0.

Our intention is to use Kripke structures as interpretation models for a logic dealing with temporal properties. To this end, the tense ordering of database states has to be reflected by the reachability relation. In modal logic oriented approaches to temporal logics, see for example [Se80], the reachability relation directly defines the tense ordering. This approach does not allow to state temporal properties of direct temporal successors, because all future worlds are handled exactly the same way.

In contrast to the pure modal logic approach, the temporal logic proposed herein makes use of the direct temporal succession as well as of the global tense ordering. To this end, the reachability relation R defines the direct temporal successions whereas the transitive closure of R fixes the tense order.

A *temporal structure* is a Kripke structure where the reachability relation R correctly reflects the temporal successions, i.e. the transitive closure of R has to be antisymmetric. A Kripke structure is called a *temporal structure*, if it satisfies the following condition :

$$R^*(\alpha,\gamma) \wedge R^*(\gamma,\alpha) \;\Rightarrow\; (\alpha = \gamma)$$

In temporal structures, the world set W and the reachability relation R define a directed acyclic graph. Of special interest are linear temporal structures of finite length. A temporal structure is called

- finite iff $|W| \,\epsilon\, I\!N$
- linear iff $|\{\gamma | R(\alpha,\gamma)\}| \leq 1$ for all $\alpha \,\epsilon\, W$

A temporal structure is *linear*, if each world has at most one direct temporal successor. Linear temporal structures are models for (finite and infinite) database state sequences. In linear temporal structures, the set W is isomorphic to an interval $(0, 1, \ldots, n)$ of the natural numbers or isomorphic to $I\!N_0$ for infinite sequences, respectively. Further, we use the notations *linear temporal structures* and *database state sequences* alternatively.

We denote the i–th database state by σ_i and the whole database state sequence by $\hat{\sigma} = \langle \sigma_0, \sigma_1, \ldots, \sigma_n \rangle$ without explicit reference to the fixed signature Σ. The reachability relation R is implicitly determined by the succession of natural numbers (i.e., $R(i,j) \iff i + 1 = j$). The tense order of the database states is expressed by the $\leq$–relation in $I\!N$. The i–th tail sequence $\langle \sigma_i, \sigma_{i+1}, \ldots \rangle$ of a state sequence $\hat{\sigma}$ is denoted by $\hat{\sigma}_i$.

The notion of substitutions in database states is straightforward extended to *global substitutions* in Kripke structures. For simplicity, we introduce global substitutions only for state sequences. A global substitution $\hat{\vartheta}$ is a sequence of substitutions ϑ_i for each database state σ_i in $\hat{\sigma}$. Global substitutions bind elements to variables as long as the objects exist in $\hat{\sigma}$. So, if a variable x of sort s is substituted in state i by a value $\vartheta_i(x\colon s)$ and this value exists in a later state j, the variable x must be substituted in σ_j by the same value :

$$\Big(\vartheta_i(x\colon s) \,\epsilon\, \sigma_j(s) \wedge i \leq j\Big) \Rightarrow \Big(\vartheta_i(x\colon s) = \vartheta_j(x\colon s)\Big)$$

Of course, this definition requires a tense independent identification mechanism for the substituted elements. For typical database objects this can be achieved by database keys or object surrogates. In the above formula, the equality sign '=' denotes identity with respect to the tense independent identification mechanism.

Now, correctness of database state sequences can be specified by dynamic constraints in a temporal logic which uses linear temporal structures for interpreting formulae.

3 Finite–Linear Temporal Logic

The linear temporal logic as introduced by, e.g., Manna and Pnueli (1981) offers powerful temporal operators like **always** or **before**, which are defined by quantification over subsequences of a linear temporal structure. At the same time, each of these temporal operators is characterized by a *temporal recursion rule* enabling a stepwise formula evaluation. An example is $\textbf{always}\,\varphi \iff \varphi \wedge \textbf{next}\,\textbf{always}\,\varphi$, where the temporal operator **always** is splitted into a current part and into a future part bounded by the nexttime operator. In our specification context, a considerable lack of this logic is the restriction to state sequences of infinite length. This restriction follows from the definition of the nexttime operator, which assumes the existence of a unique successor for each state.

The problems arising from a linear temporal logic having only one nexttime operator especially appear while using the temporal recursion rules in finite sequences. In [MP81], the recursion rules for **sometime** and **always** are

$$\begin{aligned} \textbf{always}\,\varphi &\iff \varphi \wedge \textbf{next}\,\textbf{always}\,\varphi \\ \textbf{sometime}\,\varphi &\iff \varphi \vee \textbf{next}\,\textbf{sometime}\,\varphi \end{aligned}$$

In the last state of a finite sequence, however, the following equivalences should be valid :

$$\begin{aligned} \textbf{always}\,\varphi &\iff \varphi \\ \textbf{sometime}\,\varphi &\iff \varphi \end{aligned}$$

Both together, we would have following equivalences for the nexttime operator in the last state :

$$\begin{aligned} \textbf{true} &\iff \textbf{next}\,\textbf{always}\,\varphi \\ \textbf{false} &\iff \textbf{next}\,\textbf{sometime}\,\varphi \end{aligned}$$

This looks odd, because the semantics of the nexttime operator seems to depend on the bounded subformulae — moreover, it depends on the recursion rule introducing the **next**.

In arbitrary, finite as well as infinite and linear as well as branching temporal structures, a temporal logic should contain two nexttime operators, which correspond to quantifiers over the set of immediate temporal successors. In linear structures, their semantics differs only in the case of terminating sequences in the last state. We use these nexttime operators, notated as **existsnext** and **allnext**, in our finite–linear temporal logic to distinguish between formulae requiring sequence continuation and those allowing sequence termination.

Syntax and semantics of finite–linear temporal logic formulae are defined as follows :

$(\hat\sigma, \hat\vartheta) \models \varphi'$	iff	$(\sigma_0, \vartheta_0) \models \varphi'$ for each first order logic formula φ'.
$(\hat\sigma, \hat\vartheta) \models \neg\varphi$	iff	not $(\hat\sigma, \hat\vartheta) \models \varphi$.
$(\hat\sigma, \hat\vartheta) \models \varphi \wedge \psi$	iff	$(\hat\sigma, \hat\vartheta) \models \varphi$ and $(\hat\sigma, \hat\vartheta) \models \psi$.
$(\hat\sigma, \hat\vartheta) \models \textbf{existsnext}\,\varphi$	iff	$\lvert\hat\sigma\rvert > 1$ and $(\hat\sigma_1, \hat\vartheta_1) \models \varphi$.
$(\hat\sigma, \hat\vartheta) \models \textbf{allnext}\,\varphi$	iff	$\lvert\hat\sigma\rvert > 1$ implies $(\hat\sigma_1, \hat\vartheta_1) \models \varphi$.
$(\hat\sigma, \hat\vartheta) \models \textbf{always}\,\varphi$	iff	$(\hat\sigma_i, \hat\vartheta_i) \models \varphi$ for all $0 \le i < \lvert\hat\sigma\rvert$.
$(\hat\sigma, \hat\vartheta) \models \textbf{sometime}\,\varphi$	iff	there exists i, $0 \le i < \lvert\hat\sigma\rvert$, such that $(\hat\sigma_i, \hat\vartheta_i) \models \varphi$
$(\hat\sigma, \hat\vartheta) \models \varphi\,\textbf{until}\,\psi$	iff	$(\hat\sigma_i, \hat\vartheta_i) \models \varphi$ for all $0 \le i < j$ where j is defined by $j = \min\{\{k \mid (\hat\sigma_k, \hat\vartheta_k) \models \psi\} \cup \{\lvert\hat\sigma\rvert - 1\}\}$.
$(\hat\sigma, \hat\vartheta) \models \varphi\,\textbf{before}\,\psi$	iff	there exists i, $0 \le i < j$, such that $(\hat\sigma_i, \hat\vartheta_i) \models \varphi$, where j is defined by $j = \min\{\{k \mid (\hat\sigma_k, \hat\vartheta_k) \models \psi\} \cup \{\lvert\hat\sigma\rvert - 1\}\}$.

As known from the usual first order logic quantifiers $\forall$ and $\exists$, the temporal quantifiers are paired by the following duality rules :

$$\begin{aligned}
\textbf{allnext}\,\neg\varphi &\iff \neg\,\textbf{existsnext}\,\varphi \\
\textbf{always}\,\neg\varphi &\iff \neg\,\textbf{sometime}\,\varphi \\
(\neg\varphi)\ \textbf{until}\,\psi &\iff \neg(\varphi\,\textbf{before}\,\psi)
\end{aligned}$$

The temporal operators quantifying over whole state sequences (like **always** or **until**) are related to the so–called *nexttime operators* **existsnext** and **allnext** by the following equivalences called *temporal recursion rules.* The use of the different nexttime operators in these recursion rules reflects the intended semantics for terminating state sequences and avoids the mentioned anomalies occurring in classical linear temporal logic (as can be seen by repeating the above argumentation with this rules).

$$\begin{aligned}
\textbf{always}\,\varphi &\iff \varphi \wedge \textbf{allnext}(\textbf{always}\,\varphi) \\
\textbf{sometime}\,\varphi &\iff \varphi \vee \textbf{existsnext}(\textbf{sometime}\,\varphi) \\
\varphi\,\textbf{until}\,\psi &\iff \psi \vee (\varphi \wedge \textbf{allnext}(\varphi\,\textbf{until}\,\psi)) \\
\varphi\,\textbf{before}\,\psi &\iff (\neg\psi \wedge \varphi) \vee (\neg\psi \wedge \textbf{existsnext}(\varphi\,\textbf{before}\,\psi))
\end{aligned}$$

In database specifications, finite state sequences occur as formal descriptions of finite *object life cycles.* Insertion and deletion of database objects lead to changing sort carriers during the evolution of the state sequence. An object life cycle results from changing properties of a database object between its insertion and its deletion. Such an object life cycle determines the subsequence of the global state sequence where the object exists.

Special problems occur with the extension of propositional temporal logic to a first order temporal logic. The intended semantics of a temporal formula with free variables is that this formula is valid for each *possible substitutions of the variables with database objects during their life cycle.* This leads to the notion of *reduct quantification*, where the quantification ranges over complete object life cycles instead of ranging over the object sort carrier.

First we formalize the notion of 'life cycle'. Let Θ be the set of substitutions, which maps variables of a set Y of variables to objects actual in the current state σ_0. For a given state sequence $\hat{\sigma}$ and a substitution $\vartheta \in \Theta$, the *ϑ–reduct* $\hat{\sigma}\mid_{\vartheta}$ is defined as the maximum prefix sequence of $\hat{\sigma}$ where ϑ maps the variables to the same values as in σ_0. The ϑ–reduct is always of length greater than zero, because at most the state σ_0 itself is included. If the set Y contains only one variable y, we say that the ϑ–reduct is the *life cycle* of $\vartheta(y)$. Otherwise we call it the common life cycle of the substituted objects.

Based on ϑ–reducts, we can define variable quantification for state sequences [Sa88b]. Here, we use them only for the definition of substitution independent validity having the desired semantics.

Intuitively, the substitution independent validity requires the quantification over all objects occurring during system life time. Given the set of free variables of a formula, we need the set of all substitutions of these variables having at some time actual objects as substituted values. For a given formula φ with free variables $Y \subset X$ the validity can be defined independently by varying over all appearing substitutions. Let $\Theta_Y(\hat{\sigma})$ be the set of all at some state defined substitutions ϑ of variables in Y with elements of the related sort carriers :

$$\Theta_Y(\hat{\sigma}) = \bigcup_i \left\{ \vartheta \;\middle|\; \vartheta\colon Y \to \bigcup_{s \epsilon S} \sigma_i(s) \right\}$$

With $\alpha.\vartheta$ we denote the index of the first state of a state sequence $\hat{\sigma}$ where ϑ is defined for all $y \,\epsilon\, Y$. Then $(\hat{\sigma}_{\alpha.\vartheta}\;|_{\vartheta})$ is the (first) interval of states in $\hat{\sigma}$ where ϑ is defined for all $y \,\epsilon\, Y$. The index of its last state (wrt $\hat{\sigma}$) is denoted by $\beta.\vartheta$, so that

$$\hat{\sigma}_{\vartheta} := (\hat{\sigma}_{\alpha.\vartheta}\;|_{\vartheta}) = \langle \sigma_{\alpha.\vartheta}, \sigma_{\alpha.\vartheta+1}, \ldots, \sigma_{\beta.\vartheta} \rangle$$

The *substitution independent validity* of φ is defined by

$$\hat{\sigma} \models \varphi \quad \text{iff} \quad \left(\hat{\sigma}_{\vartheta}, \hat{\vartheta}\right) \models \varphi \text{ for all } \vartheta \,\epsilon\, \Theta_Y(\hat{\sigma}) \ ,$$

which expresses the desired semantics. In informal terms, we inspect the validity of φ for all common life cycles of object combinations occurring during system life time. The natural extension of the local substitution ϑ to a global substitution $\hat{\vartheta}$ is uniquely determined by ϑ for the sequence $\hat{\sigma}_{\mu.\vartheta}|_{\vartheta}$.

4 Analysis of Temporal Logic Specifications

Finite–linear temporal logic lays the foundation of a specification calculus for database dynamics. In particular, it allows formal analysis of specifications, e.g. wrt consistency of integrity constraints.

A useful tool for that purpose is the construction of *evaluation schemes* for temporal formulae, so–called transition graphs. Using the temporal recursion rules, a temporal formula can be evaluated stepwise, i.e. state by state of the sequence. An evaluation scheme is a graph that consists of nodes labelled by those temporal formulae which occur as intermediate results during evaluation, and of edges labelled by static formulae and thus describing the correct state transitions.

Such graphs were already known for classic linear temporal logic extending propositional logic only: Manna and Wolper (1984) used them for deciding satisfiability of formulae. Here, we discuss an alternative construction which is based on formula transformation into a disjunctive normalform, and which is adapted to our finite–linear temporal logic. This construction will lead to a new satisfiability criterion for formulae in finite state sequences.

Formally, a *transition graph* $T = (G, \nu, \eta, m_0, F)$ over a signature Σ and a set X of variables consists of

- a directed graph $G = (N, E)$ with nodes N and edges E,
- a node labelling ν mapping each node to a formula in finite–linear temporal logic over Σ and X,
- an edge labelling η mapping each edge to a nontemporal formula (i.e. a formula in first order predicate logic) over Σ and X,

- a non-empty initial marking $m_0 \subseteq N$
- and a set of final nodes $F \subseteq N$.

Such a graph T can be used to accept or reject a finite state sequence $\hat{\sigma} = \langle \sigma_0, \ldots, \sigma_n \rangle$ for a given substitution ϑ by computing the nodes reached ("marked") along that sequence. The *current marking* m_T at time i for $\hat{\sigma}$ and ϑ is defined by:

$$\begin{array}{rcl} m_T(0, \hat{\sigma}, \vartheta) &=& m_0 \text{ for } 0 \leq i \leq \alpha.\vartheta \\ m_T(i+1, \hat{\sigma}, \vartheta) &=& \mathit{trans}(m_T(i, \hat{\sigma}, \vartheta), \sigma_i, \vartheta) \text{ for } \alpha.\vartheta \leq i \leq \beta.\vartheta \\ m_T(i+1, \hat{\sigma}, \vartheta) &=& m_T(i, \hat{\sigma}, \vartheta) \text{ for } \beta.\vartheta \leq i \leq n \end{array}$$

where

$$\mathrm{trans}(K, \sigma, \vartheta) = \{ l \in N \mid (\exists k \in k) : (k,l) \in E \wedge (\sigma, \vartheta) \models \eta((k,l)) \}$$

for an arbitrary subset K of nodes

The definition of trans is called *transition rule*; it formalizes that every next state in a sequence must satisfy the label of at least one edge outgoing from the current marking in order to get a new non-empty marking. Accordingly, $(\hat{\sigma}, \hat{\vartheta})$ is *accepted* by T, iff the last marking $m_T(n+1, \hat{\sigma}, \hat{\vartheta})$ contains at least one final node. Thus there must have been a path from the initial marking to a final node, such that the edge labels on that path have been valid in the corresponding states of the sequence (to be precise: of the subsequence $\hat{\sigma}_\vartheta$ from $\alpha.\vartheta$ to $\beta.\vartheta$ representing the "life cycle" of the substituting objects).

In order to evaluate a temporal formula φ in a finite state sequence $\hat{\sigma}$ by means of a transition graph T, two further conditions are needed:

- For each node $k \in N$ holds

$$\nu(k) \iff \left(\bigvee_{\substack{l \in F \\ (k,l) \in E}} (\eta((k,l)) \wedge \textbf{allnext}\, \nu(l)) \right) \vee \left(\bigvee_{\substack{l \in (N-F) \\ (k,l) \in E}} (\eta((k,l)) \wedge \textbf{existsnext}\, \nu(l)) \right)$$

- φ is represented by the disjunction of initial node labels, i.e.:

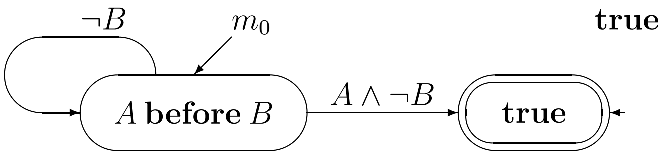
$$\varphi \iff \bigvee_{k \in m_0} \nu(k)$$

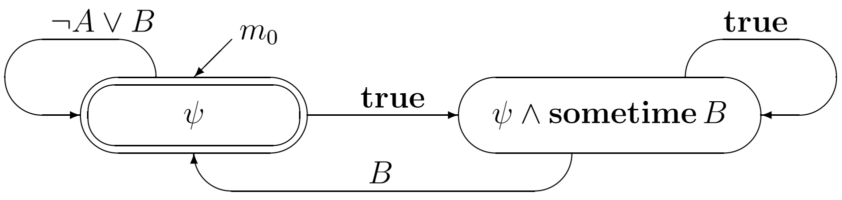
Then T is called an *evaluation scheme* for φ. This definition guarantees that the currently marked nodes are labelled with that temporal formula which remains to be evaluated in the rest of the sequence. By induction on the length of state sequences, the following relationships can be shown [Sa88a]:

- $[\hat{\sigma}, \vartheta] \models \varphi$ iff $(\hat{\sigma}, \vartheta)$ is accepted by an evaluation scheme for φ

- $\hat{\sigma} \models \varphi$ iff $\hat{\sigma}$ is accepted for arbitrary substitutions ϑ by an evaluation scheme for φ

The following figures show evaluation schemes for $\varphi \equiv A$ **before** B and $\psi \equiv$ **always**$(A \Rightarrow$ **sometime** $B)$:





Such evaluation schemes can indeed be constructed from arbitrary temporal formulae by transforming the formulae into disjunctions corresponding to the pattern above. A formula φ is in *disjunctive normalform* if it has the form

$$\bigvee_k (\zeta_k \wedge \textbf{allnext}\, \gamma_k\ [\wedge\, \textbf{existsnext}\, \delta_k])$$

where:

- each ζ_k is a conjunction of basic subformulae of φ (see below) or their negations and
- each γ_k is a conjunction of formulae γ_{k_j} or $\neg\gamma_{k_j}$ such that each γ_{k_j} is either a basic formula of φ or bound by temporal operator. (β_k by analogy.)

Basic subformulae of φ are exactly those minimal nontemporal constituents of φ which occur as operands of logical connectives outside $\forall/\exists$-quantifications.

A formula can be transformed into an equivalent disjunctive normalform by iterated application of the temporal recursion rules and of the laws of propositional logic. For example,

$$\varphi \equiv \textbf{always}(A \Rightarrow \textbf{sometime}\, B)$$

is equivalent to

$$(\neg A \vee \textbf{sometime}\, B) \wedge \textbf{allnext}(\textbf{always}(A \Rightarrow \textbf{sometime}\, B))$$

because of the recursion rule for **always**, and can further be transformed into the normalform

$$\begin{array}{llcll} & (& \neg A & \wedge & \textbf{allnext}(\textbf{always}(A \Rightarrow \textbf{sometime}\, B))) \\ \vee & (& B & \wedge & \textbf{allnext}(\textbf{always}(A \Rightarrow \textbf{sometime}\, B))) \\ \vee & (& [\textbf{true}\ \wedge] & & \textbf{allnext}(\textbf{always}(A \Rightarrow \textbf{sometime}\, B)) \\ & & & \wedge & \textbf{existsnext}(\textbf{sometime}\, B)) \end{array}$$

The quite analogous normalization procedure for infinite–linear temporal logic, which is a first–order extension of classic Manna–Pnueli–logic, has been presented in [LS87] with more details.

Now a transition graph T_φ is constructed from φ according to the following algorithm :

```
(1) start with the empty graph $G$ as $T_\varphi$;
(2) insert a node $k$ labelled with $\varphi$ into $N$;
    $m_0 := \{k\}$;
    $F := \{\}$;
(3) for each node $n \in N$ in $T_\varphi$
    do transform the node label $\nu(n)$ into disjunctive normalform $DNF(\nu(n)$;
        for each constituent $(\zeta \wedge \mathbf{allnext}\,\gamma)$ of $DNF(\nu(n))$
        do if $\nu(m) \neq \gamma$ for all $m \in F$
                then insert a new node $m$ labelled with $\gamma$ into $N$
                        $F := F \cup \{m\}$;
            endif;
            let $m$ be the node in $F$ with $\nu(m) = \gamma$;
            insert an edge $(n, m)$ with $\eta((n, m)) = \zeta$ into $E$;
        enddo;
        for each $(\zeta \wedge \mathbf{allnext}\,\gamma \wedge \mathbf{existsnext}\,\delta)$ in $DNF(\nu(n))$
        do if $\nu(m) \neq \gamma \wedge \delta$ for all $m \in N - F$
                then insert a new node $m$ with $\nu(m) = \gamma \wedge \delta$ into $N - F$;
            endif;
            let $m$ be the node in $N - F$ with $\nu(m) = \gamma \wedge \delta$;
            insert an edge $(n, m)$ with $\eta((n, m)) = \zeta$ into $E$;
        enddo;
        if  several edges lead from $n$ to the same node
            then replace them by one edge labelled with the disjunction
                    of the given labels;
            endif;
    enddo.
```

For instance, the graphs given above have resulted from this construction.

It can easily be seen that T_φ is an evaluation scheme for φ, since the initial node is labelled with φ and the edge/node labels have been derived from corresponding constituents of disjunctive normalforms. Please note that

$$\mathbf{allnext}\,\gamma \wedge \mathbf{existsnext}\,\delta \quad \Longleftrightarrow \quad \mathbf{existsnext}(\gamma \wedge \delta)$$

Such a scheme gives a decision criterion for consistency, i.e. satisfiability, of the temporal formula wrt finite state sequences. Since satisfiability of predicate logic formulae is undecidable, we have to assume that all unsatisfiable edge labels have been removed already.

- Then a temporal formula φ is satisfiable for a given substitution of its free variables in some finite state sequence if there is a path from an initial to a final node in T_φ.

Together with the construction above we have got a relative decision procedure for finite satisfiability of our temporal formulae. Despite its dependence on satisfiability in predicate logic it can be expected to be a helpful tool for analyzing temporal constraints; practical database specifications will often not utilize the full power of predicate logic for their basic subformulae.

Concerning satisfiability in infinite state sequences, the reader is referred to [LS87] extending [Wo83, MW84]. For that purpose, the difference between final and non–final nodes has to be ignored, but graphs have to be reduced by deleting infinitely unsatisfiable node labels; this can be done by a reduction algorithm which also treats the temporal structure of formulae relative to their predicate logic parts.

In other papers, we have studied other benefits resulting from the application of transition graphs. Although that work was done mainly in the context of infinite-linear temporal logic, the following statements will hold for an adaptation to finite–linear logic, too.

- Basically, the construction of transition graphs translates long-term constraints on entire state sequences into short-term conditions on single state transitions. These conditions are connected within graphical representations that describe life cycle patterns of database objects. They can help a database designer to validate his originally descriptive specification of database dynamics.

- Transition graphs form a basis for monitoring temporal constraints at runtime in the database evolution up to a current state [LS87, SL87, Sa88a]. The monitor has to compute current marking for all substitutions according to the transition rule. A constraint violation is reported as soon as a marking has become empty or does not contain a final node after a deletion (of the respective substituting objects). All paths not leading to final nodes should be deleted in order to detect errors as early as possible.

- The graphs can be used also to transform specifications of database actions, so that they respect the temporal constraints [Li88]. To this end, refined pre– and postconditions have to consider the stepwise computation and checking of markings on each specification. Then each state sequence executable by actions becomes admissible wrt the constraints, too. A related application is verification of actions against constraints [FS88].

5 Conclusions

This paper has introduced a new kind of linear temporal logic which can adequately be interpreted in infinite as well as in finite state sequences. Moreover, it allows arbitrary subformulae in first order predicate logic. Classical approaches had considered only infinite sequences for interpretation and had extended only propositional logic by temporal structures. Our proposal is motivated by the wish to specify temporal integrity constraints on the dynamic behaviour of a database. Database dynamics, however, is composed of typically finite life cycles of objects reaching from their insertion to their deletion, so that constraints have to deal with finite state sequences and with changing domains.

The central modification of classic temporal logic lies in introducing two dual versions of the **next**-operator which correspond to the universal and existential temporal quantification by **always** / **sometime**. These new operators have been integrated in such a way that the central recursion rules and the construction of transition graphs using these rules could be established again. It is expected that an axiomatization of this logic similiar [Ma82] can be given in an analogous way utilizing the new recursion rules.

An important topic of future research will be the development of a strictly object-oriented specification calculus in the framework of [SSE87], where all attributes, constraints, and actions are defined locally to the objects affected. The logic presented is an important step in that direction since it helps to express properties of finite state sequences, in particular those bound by the lifetime of an object in the database. The next step concerning adequate interpretation structures should be a decomposition of states and state sequences into single objects and their separated, but interacting life cycles.

References

[CF84] Casanova, M. A., Furtado, A. L.: On the Description of Database Transition Constraints Using Temporal Languages. *Advances in Database Theory, Vol. II (H. Gallaire et al., eds.)*, Plenum Press, New York 1984, **211–236**.

[CS87] Carmo, J., Sernadas, A.: A Temporal Logic Framework for a Layered Approach to Systems Specification and Verification. *Proc. IFIP WG 8.1 Conf. on "Temporal Aspects of Information Systems"*. Sophia–Antipolis 1987, **31–46**.

[ELG84] Ehrich, H.-D., Lipeck, U.W., Gogolla, M.: Specification, Semantics and Enforcement of Dynamic Database Constraints. *Proc. Int. Conf. on Very Large Databases*. Singapore 1984, **301–308**.

[FS88] Fiadeiro, J., Sernadas, A.: Specification and Verification of Database Dynamics. *Acta Informatica*. Vol. 25, Fasc. 6, 1988, **625–661**.

[KMS85] Khosla, S., Maibaum, T.S.E., Sadler, M.: Database Specification. *Proc. IFIP Conf. on Database Semantics (DS–1) 1985 (T.B.Steel, R.Meersmann, eds.)*. North Holland, Amsterdam 1986, **141–158**.

[Krö87] Kröger, F.: Temporal Logic of Programs. Springer–Verlag, Berlin 1987.

[Ku84] Kung, C.H.: A Temporal Framework for Database Specification and Verification. *Proc. Int. Conf. on Very Large Databases*. Singapore 1984, **91–99**.

[LEG85] Lipeck, U.W., Ehrich, H.-D., Gogolla, M.: Specifying Admissibility of Dynamic Database Behaviour Using Temporal Logic. *Proc. IFIP Work. Conf. on Theoretical and Formal Aspects in Information Systems (A.Sernadas et al., eds.)* North–Holland, Amsterdam 1985, **145–157**.

[Li88] Lipeck, U.W.: Transformation of Dynamic Integrity Constraints into Transaction Specifications. *Proc. 2nd Int. Conf. on Database Theory. (M. Gyssens et al., eds.)*LNCS 326 Springer–Verlag, Berlin 1988, **322–337**.

[LS87] Lipeck, U.W., Saake, G.: Monitoring Dynamic Integrity Constraints Based on Temporal Logic. *Information Systems.* Vol. 12, No. 3, 1987, **255–269**.

[Ma82] Manna, Z.: Verification of Sequential Programs: Temporal Axiomatization. *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. *The Correctness Problem in Computer Science (R.S.Boyer, J.S.Moore, eds.).* Academic Press, London 1981, **215–273**.

[MW84] Manna, Z., Wolper, P.: Synthesis of Communicating Processes from Temporal Logic Specifications. *ACM Transactions on Programming Languages and Systems.* Vol. 6, 1984, **68–93**.

[Sa88a] Saake, G.: Specification, Semantics and Monitoring of Object Lifecycles in Databases (in German). *Doctoral Thesis.* Techn. Universität Braunschweig 1988.

[Sa88b] Saake, G.: On First Order Temporal Logics with Changing Domains for Information System Specification. *Submitted for Publication*, 1988.

[Se80] Sernadas, A.: Temporal Aspects of Logical Procedure Definition. *Information Systems.* Vol. 5, 1980, **167–187**.

[SFNC84] Schiel, U., Furtado, A.L., Neuhold, E.J., Casanova, M.A.: Towards Multi–Level and Modular Conceptual Schema Specifications. *Information Systems.* Vol. 9, 1984, **43–57**.

[SL87] Saake, G., Lipeck, U.W.: Foundations of Temporal Integrity Monitoring. *Proc. IFIP Work. Conf. on "Temporal Aspects in Information Systems", 1987. (C. Rolland et al., eds.)*, North–Holland Publ. Co., Amsterdam 1988, **235–249**.

[SSE87] Sernadas, A., Sernadas, C., Ehrich, H.–D.: Object–Oriented Specification of Databases : An Algebraic Approach. *Proc. Int. Conf. on Very Large Databases*, 1987, **107–116**.

[Wo83] Wolper, P.: Temporal Logic Can Be More Expressive. *Information and Control 56*, 1983, **72–99**.