

ON THE DESCRIPTION OF DATABASE TRANSITION CONSTRAINTS USING
TEMPORAL LANGUAGES

Marco A. Casanova¹ and Antonio L. Furtado²

Centro Cientifico de Brasilia, IBM do Brasil, Brasil¹

Pontificia Universidade Catolica, Rio de Janeiro, Brasil²

ABSTRACT

A family of formal languages that extends classic temporal logic with mechanisms to define new modalities is described. The languages are sufficiently flexible to express a wide range of database transition constraints, that is, restrictions on what sequences of database states are meaningful. They are useful in the context of a database design methodology where the first level of specification consists of a purely declarative definition of both static and transition constraints. A sequence of results about the solvability of the decision problem of these languages is also presented that helps assess the expressive power of the languages.

INTRODUCTION

A database description, or database *schema*, consists of a set of data structure descriptions indicating how data is logically organized in the database and a set of *static constraints* capturing the semantics of the data by imposing restrictions on the allowed database states.

Considerable effort has been spent on devising formal languages tailored to the description of static constraints and on investigating their decision problem.

However, static constraints do not cover situations where restrictions on sequences of database states must be imposed, such as "salaries never decrease" and "an employee who is currently

assigned to a project cannot be fired" (i.e., he must first be disconnected from any project). Restrictions of this type are defined by *transition constraints*.

Despite their importance, almost no formal treatment of transition constraints can be found in the literature. This chapter attempts to remedy this neglect by presenting a family of formal languages to describe transition constraints.

The formal languages defined should be considered in the context of a multilevel database specification methodology introduced first in Castilho et al. [1982]. Briefly, the first level of specification corresponds to the usual assumption that a database does not include any set of application-oriented operations. Hence, transition constraints should be expressed in a purely declarative style, without referring to the way the database will be updated. The formal languages introduced in this paper were designed to be used at this level. At the second level of specification, a set of application-oriented operations, that preserve static as well as transition constraints, is defined. By convention, users' transactions must update the database only through calls to these operations. Hence, users are relieved from worrying about consistency, since transactions will necessarily preserve consistency. Thus, second level application-oriented operations offer a strategy of enforcing first-level constraints.

Returning to transition constraints, there are in principle several alternative approaches that could be used in their formalization. Perhaps the simplest one would be to use first-order languages with explicit "state" or "time" parameters acting as indices on the other terms. This approach, cast in the language of classic Temporal Logic (Rescher and Urquhart [1971]), was followed in Castilho et al. [1982] and in Clifford [1982]. We also note here that a fragment of branching time Temporal Logic was used in Mays et al. [1982] to investigate the possibility of monitoring future events in database environments. An extension of functional dependencies relating pairs of consecutive states was considered in Vianu [1983].

A second alternative would be to assume directly that the database is updated via a prespecified set of application-oriented operations, and then phrase transition constraints as properties of the operations. The drawback here is exactly that transition constraints are only implicitly specified as a consequence of assuming application-oriented operations. As we will argue in the second section, it is advantageous to have an independent description of transition constraints, defined as declaratively as possible, without any reference to how the database will be updated.

The third approach, which we follow here, would be to adopt a formal language that does not refer explicitly to states (or time) and yet is able to express restrictions on sequences of states. One such language is Temporal Logic, as described in connection with the specification and verification of concurrent programs in Pnueli [1979] Manna and Pnueli [1981] and Manna and Wolper [1981], or the specification of network protocols in Schwartz and Melliar-Smith [1981,1982]. Temporal Logic, as considered in the above references, is Propositional Calculus extended with four modalities: $\circ P$ ("P is true in the next state"), $\diamond P$ ("eventually P will be true"), $\square P$ ("henceforth P will be true") and $P \cup Q$ ("henceforth P will be true until Q is true"). A first-order-like version of Temporal Logic could also be defined by taking P to be a first-order well-formed formula (wff).

Temporal Logic proved to be suitable to express certain general properties of concurrent programs. However, as pointed out in Wolper [1981], we can easily imagine properties, particular to the concurrent program under investigation, that cannot be described using Temporal Logic. The solution proposed in Wolper [1981] consisted in expanding the expressive power of Temporal Logic by adding a mechanism to define new modalities. The mechanism is based on right-linear grammars and was developed only for the propositional version of Temporal Logic.

The situation concerning transition constraints is entirely similar. Since we want to express properties intimately related to the enterprise being modeled, and not general properties of enterprises, we cannot expect to cover all situations with a small set of modalities. Therefore, it is proposed here to express transition constraints using Temporal Logic expanded along the lines suggested in Wolper [1981]. However, unlike Wolper [1981], the full first-order-like version of the language is considered and a much more general mechanism to define modalities is adopted. This is essential to cover the wide range of transition constraints expected to arise in database modeling.

We close this introduction with a brief description of each section. A DATABASE SPECIFICATION METHODOLOGY provides a brief description of the database specification methodology underlying our approach to transition constraint description. EXTENDED TEMPORAL LOGIC presents the family of temporal languages we use to define transition constraints. THE DECISION PROBLEM FOR EXTENDED TEMPORAL LANGUAGES lists several results about the decision problem of these languages. THE EXPRESSIVE POWER OF EXTENDED TEMPORAL LANGUAGES compares the expressive power of these languages to that of other formalisms. Finally, the last section contains conclusions and directions for future work.

A DATABASE SPECIFICATION METHODOLOGY

The family of languages adopted in this paper to describe transition constraints is part of a broader database design methodology based on three levels of specification of increasing concreteness.

At the first level of specification, the database schema contains static and transition constraints defined without mentioning any set of application-oriented operations. A specification at this level serves mostly to document the intended behavior of the database. That is, it describes both the nature of the data kept in the database and the rules governing how to create and modify such data.

The formal languages described in this chapter were designed to facilitate defining transition constraints at this level of specification. That is, they do not depend on the existence of a predefined set of operations and yet their descriptive power permits defining restrictions on how data can be modified.

Proceeding to the implementation of the database, at the second level of specification a set of application-oriented operations is specified that is able to create and modify the stored data. The database schema at this second level of description includes the properties of the operations as well as their names. But no actual code for the operations is provided. The properties must be defined carefully so as to guarantee that no constraint listed in the first-level specification is violated.

The process continues by selecting a target database management system and implementing the abstract database structures and operations using the system's data description and data manipulation languages.

This specification process is governed by a notion of refinement, expressed as follows. Let D , D' and D'' be the first, second and third level specifications of the same database. Then, the following two properties must be satisfied:

- (a) programs defining application-oriented operations in D'' must satisfy all operation properties in D' ;
- (b) the set of operation properties listed in D' must guarantee all constraints defined in D (assuming that state transitions can only be brought about by the operations defined in D').

The multilevel database design methodology is justified as follows:

- (i) first-level specifications tend to be more stable since they are not affected by the addition or deletion of operations;
- (ii) second-level specifications provide an effective way of implementing constraints since database updates are encapsulated within predefined application-oriented operations (see Liskov and Zilles [1975]);
- (iii) second-level specifications are still implementation-independent since it is at the third level of specification that the database is described using the tools of a concrete database management system.

The methodology is now illustrated by describing informally a toy database about employees and projects on which they work.

Using the framework of the relational model, the first-level specification of the database is expressed as a conceptual schema S containing:

- (a) relation names: two binary relation names, EMP and ASSIGN, where EMP(n,s) is interpreted as "employee n has salary s " and ASSIGN(n,p) as "employee n works on project p ";
- (b) static constraints:
 - s1: "each employee has a unique ID number and salary";
 - s2: "every person who works on a project must be listed as an employee";
- (c) transition constraints:
 - t1: "an employee who is assigned to a project cannot be fired";
 - t2: "if an employee is fired, he cannot be rehired";
 - t3: "salaries never decrease".

This concludes the first level specification. The verbs "fire" and "rehire" as in the above were used informally for convenience. Their effect must be rephrased in terms of EMP and ASSIGN to avoid any commitment to operations at this level.

In order to pass to a second-level specification, a set of application-oriented operations is described via their properties. The set considered consists of the operations *hire*(n,s), *fire*(n), *raise-salary*(n,s), *assign-to-project*(n,p) and *release-from-project*(n,p).

We discuss briefly only the *hire* operation. The intended

effect of *hire*(*n,s*) is, of course, that *EMP*(*n,s*) becomes true. Thus, the second-level specification of the database must include the following property of *hire*:

h1: "after *hire*(*n,s*) is successfully executed, *EMP*(*n,s*) becomes true".

However, the intended effect of *hire* must be disciplined so that no constraint is violated. Thus, to preserve the static constraints, *hire*(*n,s*) should fail when *n* is an ID number already in use (since otherwise constraint *sl* would be violated). This is captured by introducing the following additional property of *hire*:

h2: "if there is *s'* such that *EMP*(*n,s'*) is true, then *hire*(*n,s*) must fail".

Consider now the transition constraint *t2*. Any sequence of operations of the form:

...*hire*(*p,e*):... *fire*(*p*):... *hire*(*p,s'*):...

violates *t2*. Hence, an additional property of *hire* must be included in the second level specification:

h3: "if *EMP*(*n,s*) was true in the past, then *hire*(*n,s'*) must fail without modifying the database".

Since *hire* does not affect any other constraint, properties *h1*, *h2* and *h3* suffice to characterize *hire* and guarantee that no constraint is ever violated.

This brief analysis of *hire* should provide a sufficient indication that the use of transition constraints requires nothing more than a disciplined way of handling users' transactions via the notion of well-defined application-oriented operations. They do not require any extra machinery than that already present in currently existing database management systems (see Tucherman et al. [1983] for a guideline on how to implement this strategy in SQL/DS).

However, it should also be clear that it may sometimes be necessary to enhance the original database structures in order to define application-oriented operations that guarantee consistency preservation. For example, to guarantee property *h3*, it may be necessary to keep an extra table of former employees.

This concludes the description of the multilevel database specification methodology. The next section presents a family of formal languages to define transition constraints in the context of first-level specifications.

EXTENDED TEMPORAL LOGIC

In this section the family of formal languages adopted to describe transition constraints is described. A brief example is also presented to motivate the work.

Let p stand for the proposition "John's salary is now 10K", q stand for "John is now an employee" and r stand for "John's salary is now less than 10K". Then, the constraint "if John's salary is now 10K and he continues to be an employee, then his salary must be at least 10K" can be rephrased as "there cannot be a sequence (S_0, \dots, S_n) of database states such that p holds in S_0 , q holds in S_1, \dots, S_{n-1} and r holds in S_n ". Or, putting it differently, no sequence of database states should satisfy some sequence of formulas of the form $pq \dots qr$ (for 0 or more q 's).

In the following we use the notation a^* to denote the set of finite words consisting of 0 or more a 's; the infinite word $aa \dots$ is not in the set denoted by a^* (see Aho and Ullman [1972]).

We now observe that the set of sequences of formulas of the form $pq \dots qr$ can be defined either by the grammar

$$G = (\{G, H\}, \{p, q, r\}, \{G \rightarrow pH, H \rightarrow qH \mid r\}, G) ,$$

or by the regular expression $p; q^*; r$. Thus, we could succinctly express the constraint in question as $\neg(p; q^*; r)$, which should be understood as "no sequence of database states should satisfy some sequence of formulas in the set denoted by $(p; q^*; r)$ ". When the set is denoted by the grammar, we introduce a new *ternary modality symbol* g and express the constraint as $\neg g(p, q, r)$, which is interpreted exactly as $\neg(p; q^*; r)$, if we understand $g(p, q, r)$ as denoting the set of all words generated from G .

To summarize, a transition constraint was defined by matching sequences of database states against sequences of formulas taken from a set denoted by a grammar (or by a regular expression).

These ideas are now developed more precisely by taking formulas either from a given first-order language, or from a given propositional language.

Let L be a first-order language and G_1, \dots, G_k be a set of grammars. The *extended temporal language* TL over L, G_1, \dots, G_k (or the *temporal extension* of L over G_1, \dots, G_k) is defined as follows. The symbols of TL are those of L plus a unary modality \circ ("next") and, for each nonterminal H of each grammar, an n_i -ary modality h , where n_i is the number of terminals of the grammar. The set of *terms* of TL is exactly the set of terms of L , and the

set of *well-formed formulas* (wffs) of TL is defined inductively as follows:

- (1) all wffs of L are wffs of TL ;
- (2) if P and Q are wffs, then $(\neg P)$ and $(P \wedge Q)$ are wffs;
- (3) if P is a wff and x is a variable of L , then $\forall xP$ is a wff;
- (4) if P is a wff, then oP is a wff;
- (5) if h is an n -ary modality and A_1, \dots, A_n are wffs then $h(A_1, \dots, A_n)$ is a wff.

Variations of these languages can be defined as follows. If we start with a propositional language L and drop rule (3), we obtain a *propositional temporal language*. If we drop rule (3), but start with a first-order language L , we obtain a *restricted temporal language*. Orthogonally, by restricting the type of grammars to be right-linear, we obtain a *right-linear temporal language*, and so on according to Chomsky's hierarchy (see, e.g., Aho and Ullman [1972]). For example, the languages considered in Wolper [1981] are what we would call right-linear propositional temporal languages.

One might as well consider regular expressions instead of right-linear grammars, which does not provide a more powerful language, but seems to produce more readable formulas. Thus, a *regular temporal extension* TL of a first-order language L is defined as follows. The symbols of TL are those of L , plus " o ", " $;$ ", " \cup ", " $*$ ". The set of wffs of TL is defined as before, except that rule (5) is replaced by:

- (5') if P and Q are wffs of TL , then $(P;Q)$, $(P \cup Q)$ and P^* are wffs of TL .

Let L be a first-order language and TL be the temporal extension of L over a given set of grammars. A *structure* of TL is a sequence $I = (I_0, I_1, \dots)$ of structures of L (the "database states") with the same domain (this restriction is somewhat important). An *assignment* of values to the variables of TL is a function v that assigns to each variable of TL a value taken from the (common) domain. We extend v to the terms of TL as for first-order languages.

If P is a first-order wff of L , I_0 is a structure of L and v is an assignment of values to the variables of L from the domain of I_0 , then we use $\models_{I_0} P[v]$ to indicate that P is valid in I_0 for the assignment v , as \models_{I_0} is standard in first-order logic (see, for example, Enderton [1972]).

Given a structure $I = (I_0, I_1, \dots)$ of TL and an assignment v of values to the variables of TL from the common domain of I , we extend the notion of validity to the wffs of TL as follows ($\models_I P[v]$ again indicates that P is valid in I for v):

- (1) if P is first-order, then $\models_I P[v]$ iff $\models_{I_0} P[v]$
- (2) if P is of the form $(\neg Q)$, then $\models_I P[v]$ iff not $\models_I Q[v]$
- (3) if P is of the form $(Q \wedge R)$ then $\models_I P[v]$ iff $\models_I Q[v]$ and $\models_I R[v]$
- (4) if P is of the form $\forall xQ$, then $\models_I P[v]$ iff $\models_I Q[u]$, for every assignment u that differs from v only on the value of x
- (5) if P is of the form $\circ Q$, then $\models_I P$ iff $\models_{I^1} Q$, where $I^1 = (I_1, I_2, \dots)$
- (6) if P is of the form $h(Q_1, \dots, Q_n)$, then $\models_I P[v]$ iff there is a word $w_{i_0} \dots w_{i_k}$ generated from H such that $\models_{I^j} Q_{i_j} [v]$, for every $j \in [0, k]$, where $I^j = (I_j, I_{j+1}, \dots)$

We note that the sentence $h(Q_1, \dots, Q_n)$ implicitly establishes a one-to-one correspondence between formulas and terminals so that Q_i corresponds to the i^{th} terminal w_{i_0} . Thus, each word $w_{i_0} \dots w_{i_k}$ corresponds to a sequence $Q_{i_0} \dots Q_{i_k}$ of wffs.

The notion of structure for propositional temporal languages is similarly defined, by making the necessary simplifications.

If TL is a regular temporal language, then we have to adapt rule (6) appropriately. We then introduce the following definitions. If P is a wff of the form $(R;Q)$, $(R \cup Q)$ or R^* , we say that P is *regular*; otherwise, we say that P is *not regular* (i.e., when P is of the form $(\neg R)$, $(R \wedge Q)$, $\forall xR$, or $\circ R$). We say that R is a *component* of P iff (i) R is P, or (ii) P is of the form $(R;Q)$, $(Q;R)$, $(R \cup Q)$, $(Q \cup R)$ or R^* , or (iii) R is a component of a component of P. Note that if P is not regular, then it has just one component, which is itself.

Let P be a regular wff. Define $A(P)$ as the set of all components of P that are nonregular wffs. Then, we may view P as a regular expression over the alphabet $A(P)$. Hence, P defines a language over $A(P)$ (i.e., a set of finite sequences of elements of $A(P)$), that we denote by $L(P)$.

Example 1

Let P be the following regular wff:

- (1) $(R;Q \cup (\forall x(T \cup U))^*)$

Then, the (immediate) components of P are

$$(2) \quad R;Q \text{ and } (\forall x(T \cup U))^*$$

and the components of components of P are

$$(3) \quad R, Q \text{ and } (\forall x(T \cup U))$$

Since $(\forall x(T \cup U))$ is not regular, T and U are not components of P . Therefore, we have that the alphabet associated with P and the language generated by P are:

$$(4) \quad A(P) = \{R, Q, (\forall x(T \cup U))\}$$

$$(5) \quad L(P) = \{\lambda, RQ, (\forall x(T \cup U)), (\forall x(T \cup U))(\forall x(T \cup U)), \dots\}$$

With the help of these definitions, we define the semantics of a regular temporal language TL just as before, except that rule (6) is replaced by:

$$(6') \quad \text{If } P \text{ is a regular formula, then } \models_I P[v] \text{ iff} \\ \text{there is a finite sequence } Q_0 \dots Q_K \text{ in } L(P) \\ \text{such that } \models_{I^j} Q_j[v], \text{ for any } j \in [0, K].$$

Finally, we say that P is *valid* ($\models P$) iff $\models_I P[v]$ for every I and v . We say that P is *satisfiable* iff there are I and v such that $\models_I P[v]$, in which case I is said to be a *model* of P . The *validity problem* for a class C of languages is the following problem: "Does there exist an algorithm (Turing machine) that takes any wff P of any language in C as input and always halts with a correct yes (P is valid) or no (P is not valid) answer?". The satisfiability problem is defined similarly.

At this point it may be observed that the languages do not contain the modalities $\diamond Q$ ("eventually Q will be true"), $\Box Q$ ("henceforth Q will always be true") and $P U Q$ ("henceforth P will always be true until Q is true"). They can be introduced by definition as follows:

$$(1) \quad \diamond Q \equiv u(\underline{\text{true}}, Q)$$

$$(2) \quad \Box Q \equiv \neg u(\underline{\text{true}}, \neg Q)$$

$$(3) \quad P U Q \equiv u(P, Q)$$

where u is induced by the following grammar

$$u = (\{U\}, \{a_1, a_2\}, \{U \rightarrow a_1 U, U \rightarrow a_2\}, U) .$$

In fact, the modality \circ (next) could also be introduced by definition. But, in view of the material to be discussed in the following section, it is not convenient to do so.

To see that these definitions agree with intuition, consider (3), for example. Given a structure $I = (I_0, I_1, \dots)$; it states that there is a sequence $P \dots PQ$ such that P is valid in I^0 through I^j and Q is valid in I^{j+1} . That is, P is valid until Q is valid. Likewise, (1) says that there is a sequence $\text{true} \dots \text{true} Q$ such that true is valid in I^0 through I^j and Q is valid in I^{j+1} . That is, there is some I^k where Q is valid, or eventually Q is valid.

Using regular temporal logic, these definitions would go as follows:

- (1) $\diamond Q \equiv \text{true}^*; Q$
- (2) $\square Q \equiv \neg(\text{true}^*; \neg Q)$
- (3) $P U Q \equiv P^*; Q$

A few transition constraints are now discussed in detail. The examples are about the same toy database used in the previous section. They are based on a first-order language L with three binary predicate symbols: EMP, ASSIGN and $>$. A wff EMP(n, s) indicates that employee n has salary s ; ASSIGN(n, p) indicates that employee n is assigned to project p ; and $s > s'$ indicates that s is greater than s' . To describe transition constraints, the regular extension TL of L is used.

Consider first the constraint "salaries never decrease". Let $S = (S_0, S_1, \dots)$ be a sequence of database states, where S_0 is the initial database state. The sequence S is unacceptable if there is $i \geq 0$ such that EMP(n, s) holds in S_i and $(\text{EMP}(n, s') \wedge s > s')$ holds in S_{i+1} . Thus, the constraint can be expressed as

- (1) $\neg \exists n \exists s (\diamond (\text{EMP}(n, s); \exists s' (\text{EMP}(n, s') \wedge s > s')))$

The wff in (1), when translated back into English, reads "it is false that there is an employee n and salaries s and s' such that eventually n has salary s in one state and salary s' less than s in the next state".

Note that, in the formalization, an employee can be fired and rehired with a lower salary. If it is understood that "salaries never decrease" rules out this situation, then an alternative formalization must be given. Let $S = (S_0, S_1, \dots)$ again be a sequence of database states. The sequence S is now unacceptable if there are $i \geq 0$ and $j > i$ such that EMP(n, s) holds in S_i and $(\text{EMP}(n, s') \wedge s > s')$ holds in S_j . Thus, the formalization now is:

$$(2) \quad \neg \exists n \exists s (\diamond (\text{EMP}(n, s) \wedge \diamond (\exists s' (\text{EMP}(n, s') \wedge s > s')))))$$

A third, and somewhat contrived, interpretation to "salaries never decrease" can also be given. It can be taken to mean that "once an employee is hired and as long as he is continuously working for the company, his current salary is never below his salary at the time he was hired". The formalization of this constraint would then forbid a sequence (S_0, S_1, \dots) where there are $i > 0$ and $j > i$ such that $\text{EMP}(n, s)$ holds in S_i , but $\neg \exists s' \text{EMP}(n, s')$ holds in S_{i-1} (i.e., n was hired at time i), $\text{EMP}(n, s')$ holds in S_{i+1} until S_j (i.e., n was an employee from $i+1$ until j) and $(\text{EMP}(n, s') \wedge s > s')$ holds in S_j (i.e., n has salary less than s at time j). Thus, the formalization is:

$$(3) \quad \exists n \exists s (\diamond (\neg \exists s' \text{EMP}(n, s'); \text{EMP}(n, s); (\exists s' \text{EMP}(n, s'))^*; \exists s' (\text{EMP}(n, s') \wedge s > s'))))$$

As another example, consider the constraint "employees who are assigned to a project cannot be fired". It can be expressed as:

$$(4) \quad \neg \exists n (\diamond (\exists p \text{ASSIGN}(n, p); \neg \exists s' \text{EMP}(n, s')))$$

Again, note that the ambiguity of natural language is avoided. In (4) it is not stated that an employee presently assigned to a project can never be fired, but that he cannot be fired without previously cancelling all his assignments to projects.

To conclude this section, we observe that triggers which indicate that some action must take place when some condition holds (see Eswaran [1976]), can also be specified as transition constraints of the form:

$$(5) \quad \square (P \Rightarrow \circ Q)$$

This states that whenever P becomes true, Q must be true in the next state. In an implementation-oriented context, triggers indicate that some action must take place when a condition P holds, the goal of the action is to make Q hold. In the present discussion, however, there is no concern with operational aspects and even less with the mechanisms (e.g. monitors) involved. Thus, triggers are specified here as transition constraints.

This concludes this section. The next section discusses the decision problem for extended temporal languages.

THE DECISION PROBLEM FOR EXTENDED TEMPORAL LANGUAGES

Several results about the decision problem for extended temporal languages are stated in this section. Besides being of interest in themselves, these results will be used in the next section to help assess the expressive power of extended temporal languages.

The Propositional Case

Two results about extended propositional temporal languages are stated in this section. Together they imply that decidability is retained if and only if at most right-linear grammars are allowed.

Theorem 1. The validity problem for the class of propositional temporal languages extended with right-linear grammar and one context-free grammar is undecidable.

Proof: The problem of deciding if a context-free grammar and a right-linear grammar generate the same language is reduced to the problem in question. Since the former problem is undecidable (cf. Aho and Ullman [1972]), the above problem is also undecidable.

Let G_1' be a context-free grammar and G_2' be a right-linear grammar. Without loss of generality, assume that G_1' and G_2' have the same set $\Sigma' = \{v_1, \dots, v_n\}$ of terminals. Construct two grammars G_i ($i=1,2$) such that:

- (i) the start symbol of G_i is S_i ;
- (ii) the set of terminals of G_i is $\Sigma = \Sigma' \cup \{v_0\}$ (assume that $v_0 \notin \Sigma'$);
- (iii) $w \in L(G_1')$ iff $wv_0 \in L(G_1)$;
- (iv) G_1 is context-free and G_2 is right-linear.

Note that conditions (iii) and (iv) are not contradictory. Then, one trivially has:

$$(1) \quad L(G_1') = L(G_2') \quad \text{iff} \quad L(G_1) = L(G_2)$$

Let L be a propositional language with $n+1$ propositional symbols p_0, \dots, p_n . Let TL be the extension of L via G_1 and G_2 . Let s_i ($i=1,2$) be the $(n+1)$ -ary modality corresponding to S_i . We show that

$$(2) \quad \models s_1(p_0, \dots, p_n) \equiv s_2(p_0, \dots, p_n) \quad \text{iff} \quad L(G_1) = L(G_2) .$$

This suffices to establish our result since, by (1) and (2), the problem of testing if a context-free grammar G_1' and a right-linear grammar G_2' generate the same language is reduced to testing the validity of a wff of a language in the class of propositional temporal languages extended with a right-linear and a context-free grammar.

(\Leftarrow) Assume that $L(G_1) = L(G_2)$. Then, the result follows trivially, by definition of validity.

(\Rightarrow) Assume that $\models s_1(p_0, \dots, p_n) \equiv s_2(p_0, \dots, p_n)$ that is, assume that

(3) for any structure I of TL , $\models_I s_1(p_0, \dots, p_n)$ iff $\models_I s_2(p_0, \dots, p_n)$.

We first show that $L(G_1) \subseteq L(G_2)$. Let $w = v_{i_0} v_{i_1} \dots v_{i_\ell} v_0 \in L(G_1)$.

Let $\bar{w} = p_{i_0} p_{i_1} \dots p_{i_\ell} p_0$ be the sequence of propositional symbols corresponding to w . Construct a structure $I = (I_0, I_1, \dots)$ of TL as follows:

- (i) for each $j \in [0, \ell]$, $I_j(p_{i_j}) = \underline{\text{true}}$ and $I_j(q) = \underline{\text{false}}$, for any propositional symbol q other than p_{i_j} ;
- (ii) $I_{\ell+1}(p_0) = \underline{\text{true}}$ and $I_{\ell+1}(q) = \underline{\text{false}}$, for any propositional symbol q other than p_0 ;
- (iii) for each $j > \ell+1$, $I_j(q) = \underline{\text{false}}$, for any propositional symbol q .

Now, by construction of I , $\models_I s_1(p_0, \dots, p_n)$. Hence, by (3), $\models_I s_2(p_0, \dots, p_n)$. Therefore, there is $u = v_{k_0} \dots v_{k_m} v_0 \in L(G_2)$ such that $I_j(p_{k_j}) = \underline{\text{true}}$, for any $j \in [0, m]$, and $I_{m+1}(p_0) = \underline{\text{true}}$. But, by construction of I , $I_j(p_{k_j}) = \underline{\text{true}}$ iff $k_j = i_j$, for any $j \in [0, \min(\ell, m)]$, and $I_{m+1}(p_0) = \underline{\text{true}}$ iff $\ell = m$. Hence, we may conclude that $w = u$. Therefore, $w \in L(G_2)$. Thus, $L(G_1) \subseteq L(G_2)$. Likewise, it can be proven that $L(G_2) \subseteq L(G_1)$, which permits one to conclude that $L(G_1) = L(G_2)$, as was to be shown.

Theorem 2. The satisfiability and the validity problems for right-linear propositional temporal languages are decidable in exponential time.

The proof of this theorem uses an adaptation of the tableaux method of classic Temporal Logic (see Rescher and Urquhart [1971]). Since it is quite long, the reader is referred to Wolper [1981] and to Casanova and Furtado [1982] for a detailed proof.

This concludes the discussion on the decision problem for extended propositional temporal languages.

The General Case

In this section it is proven that the validity problem of extended temporal languages is not partially solvable. Actually, it is shown that even the special case of regular temporal languages is not partially solvable.

To prove the validity problem of regular temporal languages is not partially solvable, it is shown that for any fixed regular program schema r there is a wff P_r such that P_r is valid iff r never halts for any interpretation and any initial state. Since the divergence problem for regular program schemes is not partially solvable, then the validity problem of regular temporal languages is not partially solvable. It is also shown that, as a consequence, there is no consistent and complete axiom system for regular temporal languages.

A brief discussion of regular program schemes is given first. Let L be a first-order language. The set R of *regular program schemes over L* (or, simply, *programs*) is defined inductively as follows (see Harel [1979]):

- (1) if x is a variable, t a term and B an atomic wff of L , then $x := t$ and $B?$ are programs called, respectively, an *assignment* and a *test*;
- (2) if r and s are programs, then r^* , $r \cup s$ and $r; s$ are also programs.

An *interpretation* for a program r over L is simply an interpretation A for L . A *state* for L and A is an assignment of values from the domain of A to the variables of L . The *universe* U of L and A is the set of all states of L and A .

Given the universe U of L and A , we define a function $m_A: R \rightarrow 2^{U^2}$ associating a relation $m_A(r) \subseteq U^2$ with each program $r \in R$. The function m_A is defined inductively as follows:

- (1) $m_A(x := t) = \{(v, v') \in U^2 / v'(x) = \bar{v}(t) \text{ and } v'(y) = v(y), \text{ for any variable other than } x\}$

Note: \bar{v} denotes the extension of v to the terms of L , using A .

- (2) $m_A(B?) = \{(v, v) \in U^2 / \models_A B[v]\}$
- (3) $m_A(r^*) = (m_A(r))^*$ —the reflexive and transitive closure of $m_A(r)$

- (4) $m_A(r \cup s) = m_A(r) \cup m_A(s)$ — the union of $m_A(r)$ and $m_A(s)$
- (5) $m_A(r; s) = m_A(r) \cdot m_A(s)$ — the composition of $m_A(r)$ and $m_A(s)$.

Now, a program r over L is said to *diverge* under interpretation A iff $m_A(r) = \emptyset$. The *divergence problem* for regular program schemes is: "Does there exist an algorithm that takes any program r as input and always halts with a correct "YES" (r diverges for every interpretation) or "NO" (r does not diverge for every interpretation) answer".

Lemma 4. The divergence problem for regular schemes is not partially solvable.

Proof (sketch): There is a straightforward reduction of the divergence problem in question to the divergence problem for program schemes, which is not partially solvable (see Manna [1974]).

It is now shown that the validity problem for regular temporal languages is also not partially solvable by reducing the divergence problem to it. The validity problem is: "Does there exist an algorithm that takes any wff P of any regular temporal language as input and always halts with a correct "YES" (P is valid for every interpretation) or "NO" (P is not valid for every interpretation) answer".

Theorem 3. The validity problem for regular temporal languages is not partially solvable.

Proof: We reduce the divergence problem for regular program schemes to the validity problem in question. Since by Lemma 4, the former problem is not partially solvable, the latter problem is also not partially solvable. Given a regular program scheme r , construct a wff $\neg P_r$ of a regular temporal language such that $\neg P_r$ is valid iff r diverges for any interpretation or, equivalently, P_r is satisfiable iff there is an interpretation A of r such that $m_A(r) \neq \emptyset$.

Let x_1, \dots, x_k be the variables occurring in r , let f_1, \dots, f_ℓ be the function symbols occurring in r and let p_1, \dots, p_m be the predicate symbols occurring in r . Let L be the first-order language whose nonlogical symbols are exactly $f_1, \dots, f_\ell, p_1, \dots, p_m$ and whose variables are x_1, \dots, x_k . Then r can be considered as a program over L . Let L' be another first-order language whose nonlogical symbols are $f_1, \dots, f_\ell, p_1, \dots, p_m$, plus a set of constants c_1, \dots, c_k . Interpret c_i as corresponding to x_i in the following sense. Given a structure A of L and an assignment v of values from the domain of A to x_1, \dots, x_k , denote by A_v the structure of L' such that $A_v(f_i) = A(f_i)$, $1 \leq i \leq \ell$, $A_v(p_i) = A(p_i)$, $1 \leq i \leq m$, and $A_v(c_i) = v(x_i)$, $1 \leq i \leq k$. The wff P_r corresponding to r is a

wff in the regular temporal language TL' extending L' .

Before constructing P_r , some auxiliary wffs are introduced. Let A be the wff

$$\bigwedge_{j=1}^k \exists y_j (c_j = y_j \wedge \circ c_j = y_j)$$

This formula is satisfiable by a structure $I = (I_0, I_1, \dots)$ of TL' iff the value of c_j , $1 \leq j \leq k$, is the same in I_0 and I_1 . Likewise, let A_i be the wff below.

$$\bigwedge_{\substack{j=1 \\ j \neq i}}^k \exists y_j (c_j = y_j \wedge \circ c_j = y_j)$$

which is satisfiable by I iff the value of c_j , $1 \leq j \leq k$ and $j \neq i$, is the same in I_0 and I_1 . Let B be the wff

$$\bigwedge_{j=1}^{\ell} \forall \bar{z}_j \forall y_j (f_j(\bar{z}_j) = y_j \Leftrightarrow \circ f_j(\bar{z}_j) = y_j)$$

which is satisfiable by I iff the value of f_j , $1 \leq j \leq \ell$, is the same in I_0 and I_1 (note that I_0 and I_1 have, by definition, the same domain). Finally, let C be the wff:

$$\bigwedge_{i=1}^m \forall \bar{z}_i (p_i(\bar{z}_i) \Leftrightarrow \circ p_i(\bar{z}_i))$$

which is satisfiable by I iff the value of p_j , $1 \leq j \leq m$, is the same in I_0 and I_1 (again the fact that I_0 and I_1 have the same domain is important).

If t is a term of TL and x_1, \dots, x_k are variables occurring in t and c_1, \dots, c_k are terms of TL , then $t[c_1/x_1, \dots, c_k/x_k]$ denotes the term obtained by replacing every occurrence of x_i by c_i , $1 \leq i \leq k$. The expression $Q[c_1/x_1, \dots, c_k/x_k]$ is used with similar meaning, if Q is a wff of TL .

P_r is now defined by induction on the structure of r :

(a) if r is $x_i := t$, then P_r is

$$A_i \wedge B \wedge C \wedge \exists y_i (y_i = t[c_1/x_1, \dots, c_k/x_k] \wedge \circ y_i = c_i)$$

which is satisfiable in $I = (I_0, I_1, \dots)$ iff the value of c_i in I_1 is equal to the value of $t[c_1/x_1, \dots, c_k/x_k]$ in I_0 , and the value of all other symbols are the same in I_0 and I_1 .

(b) If r is $Q?$, then P_r is $A \wedge B \wedge C \wedge Q[c_1/x_1, \dots, c_k/x_k]$

which is satisfiable by I iff the value of all symbols are the same in I_0 and I_1 and Q is satisfiable in I_0 .

(c) If r is $p \cup q$, $p; q$ or p^* , then P_r is $P_p \cup P_q$, $P_p; P_q$ or P_p^* , respectively.

It is now necessary to prove that:

(*) P_r is satisfiable iff $m_A(r) \neq \emptyset$, for some structure A of L

Before proceeding to prove (*), we observe that a regular program scheme r can be viewed as a regular expression over the alphabet of tests and assignments. Likewise P_r can be viewed as a regular expression over the alphabet of wffs of the form given by (a) and (b) above. Thus, r and P_r can be viewed as denoting sets of finite words in the appropriate alphabet.

(\Leftarrow): Suppose that there is a structure A of L such that $m_A(r) \neq \emptyset$. Let U be the universe of A and L . Then, since $m_A(r) \neq \emptyset$, there is a word $s_0 \dots s_n$ in the set denoted by r such that s_i is either an assignment or a test. Moreover, there is a sequence $\bar{v} = (v_0, \dots, v_{n+1})$ in U such that $(v_i, v_{i+1}) \in m_A(s_i)$, $0 \leq i \leq n$. We also have, by construction of P_r , that $P_{s_0} \dots P_{s_n}$ is in the set denoted by P_r . Now, let $I = (A_{v_0}, \dots, A_{v_{n+1}})$. Then, by construction of P_r and the basic property of \bar{v} , we have that $\models_{I_i} P_{s_i}[u]$, where I^i is the subsequence of I starting on I_i and u is any fixed assignment of values to the variables of L' (u is actually irrelevant since P_{s_i} is closed). Hence, P_r is satisfiable.

(\Rightarrow): Suppose that P_r is satisfiable. Then, there is a word $P_0 \dots P_n$ in the set denoted by P_r , a sequence $I = (I_0, \dots)$ of structures of L' and an assignment of values to the variables of L' such that $\models_{I_i} P_i[u]$, $0 \leq i \leq n$. By construction of P_r , for any $i, j \in [0, n]$, the structures I_i and I_j are equal on the values of $f_1, \dots, f_\ell, p_1, \dots, p_m$. Thus, I induces a structure A of L . Moreover, I also induces a sequence $\bar{v} = (v_0, v_1, \dots)$ of assignments of values to the variables x_1, \dots, x_k of L , where $v_i(x_j) = I_i(c_j)$, for $i \geq 0$ and $1 \leq j \leq k$. But, by construction of P_r , the word $P_0 \dots P_n$ induces a word $s_0 \dots s_n$ in the set denoted by r such that $(v_i, v_{i+1}) \in m_A(s_i)$, $0 \leq i \leq n$. Hence, $(v_0, v_{n+1}) \in m_A(r)$ and, so, $m_A(r) \neq \emptyset$, as was to be shown.

Thus, for any given regular program scheme r , a wff P_r of a regular temporal language may be constructed such that P_r is satisfiable iff there is an interpretation A of r such that $m_A(r) \neq \emptyset$.

Or, equivalently, $\neg P_r$ is valid iff r diverges for any interpretation A .

Theorem 3 has one important consequence as given by the following corollary.

Corollary 1. There is no consistent and complete axiom system for regular temporal languages.

Proof: If there were a consistent and complete axiom system, then the set of valid wffs would be recursively enumerable, but this contradicts Theorem 3.

THE EXPRESSIVE POWER OF EXTENDED TEMPORAL LANGUAGES

The family of languages discussed in previous sections was introduced on the grounds that they can adequately express transition constraints. This section contributes to substantiate this claim by investigating the expressive power of different classes of temporal languages and by comparing the use of extended temporal languages with a first-order approach.

The Expressive Power of Classes of Temporal Languages

Given that extended temporal languages are based on grammars, one is naturally inclined to conjecture that Chomsky's hierarchy of grammars (see, e.g., Aho et al. [1969]) directly induces a hierarchy of languages. Although this conjecture seems to be true, it turned out to be quite difficult to prove. The reason is simple. In addition to grammars, any proof of this conjecture must deal with logical connectives and nesting of modalities, not to mention quantifiers, which greatly complicates matters.

In spite of this negative remark, for the propositional case, one can easily prove that right-linear temporal languages are indeed less expressive than context-free temporal languages using the results in the previous section.

Given two families of propositional temporal languages, L and L' , we say that L is *at least as expressive as* L' (written $L' \leq L$) iff there is a Turing machine TM that accepts as input any wff P' of any language L' in L' and outputs a wff P of a language L of L such that P and P' have the same set of models. (Note that L and L' must have the same set of propositional symbols so that it makes sense to compare models of P to models of P' .) Intuitively, any definition in L' of a transition constraint (formalized as P') can be mechanically translated (via TM) into a definition in L (formalized as P).

We say that L is as expressive as L' iff $L' \leq L$ and $L \leq L'$ (written $L = L'$). We also say that L' is less expressive than L (written $L' < L$) iff $L' \leq L$ and $L \neq L'$.

The following propositions are direct consequences of the above definition and results in the previous section.

Proposition 1. Let L and L' be two families of extended propositional temporal languages. Suppose that $L' \leq L$. Then, if the validity problem for L is decidable, then the validity problem for L' is also decidable.

Proof: Given any wff P' of a language in L' , the Turing machine TM produces a wff P of a language in L such that P and P' have the same set of models. Hence P is valid iff P' is. Therefore, if the validity problem of L is decidable, so is the validity problem of L' .

Proposition 2. The family L' of right-linear propositional temporal languages is less expressive than the family L of context-free propositional temporal languages.

Proof: By definition, it trivially follows that $L' \leq L$. Suppose that $L \leq L'$. Then, by Proposition 1 and Theorem 2, the validity problem for L would be decidable, which contradicts Theorem 1. Hence it is not true that $L \leq L'$. Therefore, $L' < L$.

To conclude this subsection, a fairly natural example of a transition constraint that can be expressed using context-free temporal languages, but not right-linear temporal languages, is given.

Consider the problem of controlling the refereeing process of papers submitted to a conference. Assume for simplicity that a paper is refereed by just one person. Whenever a paper is received, an entry is made in a database and the paper is sent out to be refereed. When a referee report not previously received arrives, an entry is also made in the database. The number of papers submitted is not known a priori. However there must be as many referee reports received as there were papers submitted.

This last sentence expresses a transition constraint that must be satisfied by the complete history of database states. That is, after all papers are received and refereed, the sequence of states must satisfy the constraint Q , where:

- Q is $h(P,R)$;
- P is a propositional symbol interpreted as the submission of a paper;

- R is another propositional symbol interpreted as the arrival of a referee's report not previously received;
- h is a binary modality associated with the start symbol of the grammar $(\{H\}, \{p, r\}, \{H \rightarrow pr \mid prH \mid pHr\}, H)$, that generates sentences w such that w has as many p's as r's and any prefix of w has at least as many p's as r's.

It follows from classical results in formal language theory (see Aho et al. [1969]) that the language generated by H is not right-linear. Thus, there is no way of finding a wff Q' equivalent to Q such that Q' can be expressed in a right-linear propositional temporal language.

Comparison with First-Order Languages

In this subsection extended temporal languages are compared briefly with first-order languages on three different aspects: syntax, effectiveness and semantics.

Since the first-order approach adopted is quite similar to classic temporal logic (as described in Rescher and Urquhart [1971]), this subsection can also be understood as a comparison between extended temporal languages and classic temporal logic.

To illustrate the discussion consider again the constraints:

- s1. "each employee has a unique ID number and salary"
- t1. "an employee who is working on a project cannot be fired"
- t3. "salaries never decrease"

In a purely first-order approach, they could be formalized using a first-order language with the following nonlogical symbols:

- EMP, a ternary predicate symbol, with EMP(n,s,t) interpreted as "employee n has salary s in state t";
- ASSIGN, a ternary predicate symbol, with ASSIGN(n,p,t) interpreted as "employee n is assigned to project p in state t";
- AC, a binary predicate symbol, with AC(t,u) interpreted as "state u lies in the future of (is accessible from) state t";
- SUC, a binary predicate symbol, with SUC(t,u) interpreted as "state u is the immediate successor of state t".

The constraints could then be formalized as:

- s1. $\forall n \forall s \forall s' \forall t (EMP(n,s,t) \wedge EMP(n,s',t) \Rightarrow s = s')$
- t1. $\neg \exists n \exists t \exists u (\exists p ASSIGN(n,p,t) \wedge SUC(t,u) \wedge \neg \exists s EMP(n,s,u))$
- t3. $\neg \exists n \exists s \exists t \exists u (EMP(n,s,t) \wedge AC(t,u) \wedge \exists s' (EMP(n,s',u) \wedge s > s'))$.

In order to facilitate the comparison, the temporal logic formalization of these constraints are repeated below:

- s1. $\Box \forall n \forall s \forall s' (EMP(n,s) \wedge EMP(n,s') \Rightarrow s = s')$
 t1. $\neg \exists n (\diamond (\exists p \text{ ASSIGN}(n,p) \wedge \circ \neg \exists s \text{ EMP}(n,s)))$
 t3. $\neg \exists n \exists s (\diamond (EMP(n,s) \wedge \diamond (\exists s' \text{ EMP}(n,s') \wedge s > s'))))$.

Let us first compare the two approaches from the syntactical point of view.

Corresponding sentences in the two sets above have approximately the same structure. However, the first-order formalization explicitly refers to states and relationships between states in order to capture transition constraints, which forces the use of an extra entry in EMP and ASSIGN. By contrast, all this extra machinery is hidden by the special syntax of temporal languages. There is no need to consider an extra column in EMP and ASSIGN to express transition constraints. However, there is a need to introduce the additional symbols \circ , \diamond , and \Box .

Hence, it is our opinion that temporal languages can be considered better suited to describe transition constraints since they have a syntax tuned to describe state transitions, which avoids modifying the symbols used to describe database structures and which also makes it unnecessary to introduce special predicate symbols to capture relationships between states. (A similar remark appears in Manna and Pnueli [1981] in the context of program verification).

Consider now the validity problem of extended temporal languages vis-a-vis the validity problem of first-order languages. By Theorem 3 of the previous section the first problem is not even partially decidable, whereas the second one, although undecidable, is partially decidable. Therefore, one may conclude that extended temporal languages cannot be replaced by first-order languages without losing expressive power (as otherwise, by an argument similar to that in the previous subsection, one would be forced to conclude that the validity problem of extended temporal languages is partially decidable).

The last, and the most delicate aspect concerns the semantics of the two approaches. Let T be the extended temporal language and F be the first-order language used to describe a given database. Assume that F has two binary predicate symbols, SUC and AC, with the same intended meaning as previously considered.

Let I_T be a structure of T. Then I_T is a linear sequence of database states corresponding to a possible history of the database.

I_T clearly indicates, for any given database state, which state is next and which states lie in its future. This is fixed by definition and cannot be changed without deep consequences to the theory.

Let I_F be a structure of F . Let ac and suc be the interpretations of AC and SUC . To conform with the intended meaning of AC and SUC , the following restriction must be satisfied:

(*) ac must be the reflexive and transitive closure of suc .

However, the restriction in (*) is not first-order definable (see Carvalho and Veloso [1979] for an interesting general discussion involving this point). Hence, there is no hope of finding a first-order theory whose language is F and whose models would always assign the intended interpretation to SUC and AC . Therefore, the apparent simplicity of what was termed the first-order approach is lost.

On the other hand, if AC and SUC are open to interpretation, there is room for defining families of first-order-like languages that differ on what is assumed about the interpretation of these symbols. For example, if the interpretation of AC must always be a tree-like structure, then one would have a family of languages based on the notion of branching time (see Rescher and Urquhart [1971]).

To achieve the same effect using extended temporal languages, the definition of structure (and validity) would have to be changed, which is a deep modification in the development of the formalism.

In conclusion, one may say that the syntax of extended temporal languages clearly separates dynamic aspects (state transitions) from static aspects (what is asserted about each state), whereas a first-order approach permits different assumptions to be captured about the structure of the set of states with far greater flexibility than the extended temporal languages framework.

Time in Extended Temporal Languages

One final word should be said about the role of time in extended temporal languages. To begin with, an unqualified reference to time is ambiguous, since the time a (real-world) event occurred may be distinct from the time the event was recorded in the database. In this case, there is an "actuality lag" (Bubenko [1977]) between data in the database and real-world events. However, if one considers that events take place through the database, this lag vanishes. For example, one may consider that a reservation occurs when it is accepted by the database system.

To capture the time an event was installed in the database, assume that the extended temporal language L used to describe the database has a distinguished constant, τ , whose intended interpretation in a given database state is the time the state was created. This constant τ can then be used to formulate constraints. No special assumption is made about the duration of a transition in the extended temporal language framework, except that transitions are not instantaneous. This imposes a restriction on the interpretations given to τ in a sequence of database states, which is expressed as the following axiom:

$$(*) \quad \square \forall t (\tau = t \rightarrow \circ \tau > t)$$

A comprehensive treatment of time in databases can be found in Bolour et al. [1982] and a brief survey of the problems involved appears in Ariav et al. [1983].

CONCLUSION

A family of temporal languages flexible enough to express complex transition constraints was defined in a natural way. The expressive power of the languages is largely due to the avoidance of a fixed set of modalities in favor of a mechanism to define new modalities. Since the mechanism is based on grammars, Chomsky's hierarchy directly induces a classification of the languages.

This classification was used to obtain results about the decision problem of these languages. In particular, it was shown that the validity problem for right-linear (or regular) propositional temporal languages is decidable, but the validity problem for context-free propositional temporal languages is not. It was also shown that the validity problem for regular (first-order) temporal languages was not even partially decidable.

Decidability theorems were in turn used to obtain results about the expressiveness of these languages. It was shown that right-linear propositional temporal languages are less expressive than context-free propositional temporal languages in a precise sense. This result was substantiated with a natural example taken from database experience. Finally, it was observed that extended temporal languages are necessarily richer than a first-order approach to the description of transition constraints.

REFERENCES

1. Aho, A. V., Hopcroft, J. E. and Ullman, J. D. [1969] *Formal Languages and their Relation to Automata*, Addison-Wesley, Reading, Mass.
2. Aho, A. V. and Ullman, J. D. [1972] *The Theory of Parsing, Translation and Compiling*, Prentice-Hall, Englewood Cliffs, New Jersey, 1972.
3. Ariav, G., Clifford, J. and Jarke, M. [1983] "Panel on Time and Databases", *ACM SIGMOD RECORD*, 13 (4) (1983) 243-245.
4. Bolour, A., Anderson, T. L., Dekeyser, L. J. and Wong, N.K.T. [1982] "The Role of Time in Information Processing: A Survey," *ACM SIGMOD RECORD* 12 (3) (April 1982) 27-50.
5. Bubenko, J. [1977] "The Temporal Dimension in Information Modeling", in *Architecture and Models in Database Management Systems*, Nijssen, G. (Ed.), North-Holland, Amsterdam.
6. Carvalho, R. L. and Veloso, P.A.S. [1979] "Towards a Logic of Limited Perception", *Proceedings of the Third Brazilian Conference on Mathematical Logic*, Recife, Brazil (1980) 147-159.
7. Casanova, M. A. and Furtado, A. L. [1982] "A Family of Temporal Languages for the Description of Transition Constraints", *Proceedings of the 3rd Workshop on Logical Bases for Databases*, Toulouse, France (Dec. 1982).
8. Castilho, J.M.V., Casanova, M. A. and Furtado, A. I. [1982] "A Temporal Framework for Database Specifications", *Proceedings of the 8th International Conference on Very Large Databases*, Mexico City, Mexico (Sept. 1982) 280-291.
9. Clifford, J. [1982] "A Model for Historical Databases", *Proceedings of the 3rd Workshop on Logical Bases for Databases*, Toulouse, France (Dec. 1982).
10. Enderton, H. B. [1972] *A Mathematical Introduction to Logic*, Academic Press, New York, 1972.
11. Eswaran, K. P. [1976] "Specification, Implementation and Interaction of a Trigger Subsystem in an Integral Database System", *IBM Research Report RJ1820*, San Jose, CA (August 1976).
12. Griethuysen, J. J. (Ed.) [1982] "Concepts and Terminology for the Conceptual Schema and the Information Base", Report from the ISO TG97/SC5,WG3 group.

13. Harel, D. [1979] *First-Order Dynamic Logic*, Lecture Notes in Computer Science, Vol. 68, Goos, G. and Hartmanis, J. (Eds.), Springer-Verlag, Berlin.
14. Liskov, B. and Zilles, S. [1975] "Specification Techniques for Data Abstractions", *IEEE Trans. on Software Engineering*, SE-1 (1975) 7-19.
15. Manna, Z. [1974] *Mathematical Theory of Computation*, McGraw-Hill, New York.
16. Manna, Z. and Pnueli, A. [1981] "Verification of Concurrent Programs, Part I: The Temporal Framework", *Rep. STAN-CS-81-872*, Dept. of Computer Science, Stanford University, Stanford, CA (June 1981).
17. Manna, Z. and Wolper, P. [1981] "Synthesis of Communicating Processes from Temporal Logic Specifications", *Report STAN-CS-81-872*, Dept. of Computer Science, Stanford University, Stanford, CA (September 1981).
18. Mays, E., Webber, B. and Joshi, A. [1982] "Temporal Logic for Competent Database Monitors", *Proceedings of the 3rd Workshop on Logical Bases for Databases*, Toulouse, France (Dec. 1982).
19. Pnueli, A. [1979] "A Temporal Logic of Programs", *Proceedings of the 18th Foundations of Computer Science Conference*, Providence, RI (November 1979) 46-57.
20. Rescher, N. and Urquhart, A. [1971] *Temporal Logic*, Springer-Verlag, Wien (1971).
21. Schwartz, R. L. and Melliar-Smith, P.M. [1981] "Temporal Logic Specification of Distributed Systems", *Proceedings of the 2nd International Conference on Distributed Systems*, Paris, France (April 1981) 1-9.
22. Schwartz, R.L. and Melliar-Smith, P.M. [1982] "From State Machines to Temporal Logic: Specification Methods for Protocol Standards", *IEEE Trans. on Communications*, Vol. 30 (12) 2486-96.
23. Tucherman, L., Furtado, A.L. and Casanova, M.A. [1983] "A Pragmatic Approach to Structured Database Design" (accepted for the 9th Int'l. Conf. on Very Large Databases, Florence, Italy, (October 1983)).
24. Vianu, V. [1983] "Dynamic Constraints and Database Evolution", *Proceedings of 2nd ACM SIGACT-SIGMOD Symp. on Principles of Database Systems*, Atlanta, GA (March 1983) 389-399.
25. Wolper, P. [1981] "Temporal Logic can be more Expressive", *Proc. Foundations of Comp. Sci. Conf.* (1981) 340-348.