ON VERIFICATION OF DATABASE TEMPORAL CONSTRAINTS

C H Kung *

Dept of Computer Science
The Norwegian Institute of Technology
Trondheim NORWAY

ABSTRACT A database specification consists of static and temporal constraints and a set of database operation descriptions A database is viewed as a dynamic object and a sequence of database states constitutes an evolution of the database A formal method for verifying database specifications is proposed The method checks if the static constraints are consistent, analyses the database operation descriptions to ensure that each operation can ever be executed, and finally, it verifies that each permissible sequence of operations satisfies all the temporal constraints

KEYWORDS AND PHRASES Software Engineering, Databases, Semantic Integrity, Temporal Dimension, Formal Verification of Specifications

1 INTRODUCTION

During the last decade, a large number of data models and database definition languages have been proposed Most of them aim at capturing more of the semantics of reality [11] [16] and establishing a formal foundation of modelling [36]

Despite this development in data modelling, the qualitative aspects such as model validity, consistency and reliability has yet received very little attention from the research community Although this problem has been observed by Bubenko in 1977 [5], the progress has been relatively slow In particular, few results have been published on formal verification of database specifications

By formal verification of a database specification, we mean a formal process which ensures that the various components of the specification are free from conflict That is, the database specification must be consistent

* On leave from Academia Sinica, PRC

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery To copy otherwise, or to republish, requires a fee and/or specific permission

© 1985 ACM 0-89791-160-1/85/005/0169 $00 75

There are two important aspects concerning formal verification of database specifications First, the consistency of a database specificaiton is a necessary condition for a database to be regarded as a representation of some perceived world According to Nicolas and Gallaire [30], "The world ought to be a (mathematical) model of a first order theory T (which is the relational database schema) How can this be ensured? In fact there is no way of ensuring this The only thing that can be done is to verify the consistency of the theory T If it is not consistent, no model of it exists and so, the world cannot be one model either " Also in [4], it is argued that "a database exhibits semantic integrity only if all specified constraints are satisfied If any two of the large number of constraints are inconsistent, no database can exhibit semantic integrity "

Second, it is shown in [12] and [7], that faults which are introduced during database specification and design have a major impact on systems development effort On the other hand, if a database specification is formally verified before implementing it, then certain specification errors might be removed prior to the operational phase, which might reduce the total systems cost

In this paper, we propose a temporal framework for database specification and verification A database specification consists of three parts

1) A set of static constraints which are specified as first order logic sentences The static constraints must be satisfied by every legal database state

2) A set of temporal constraints which are specified as temporal assertions of a modal system The temporal constraints have the property that every permissible sequence of operations must satisfy all the temporal constraints

3) A set of operation descriptions each of which specifies a precondition and a postcondition of the operation Moreover, an operation description may contain a temporal assertion which specifies the desirable database history for applying the operation In agreement with [8] [15], we assume that database transactions are specified in terms of database operations

Correspondingly the consistency checking method also consists of three parts

1) Consistency checking of static constraints This can be done by using either resolution method [22] [33] or tableaux approah [23]

2) Analysis of operation descriptions which analyses each operation description to see whether the operation can ever be executed in the future database That is, we examine if there is a legal database state in which the precondition of the operation holds and the application of the operation yields a legal database state (Conf [14]) The analysis also checks for undesirable side effects The result of the analysis is a state transition diagram Each state of the transition diagram can be interpreted as an abstract database state in which all the static constraints are true

3) Finally, we test the consistency of the temporal constraints as follows First we transform the transition diagram into a family of pushdown automata by taking into account the temporal assertions of the operations Secondly, we use the transition diagram to generate a set of test sequences each of which is a sequence of operations Thirdly we use the pushdown automata to analyse the acceptance of the test sequences Intuitively, a sequence of operations is allowed to be executed in the future database if it is accepted by one of the pushdown automata Finally, the consistency of the temporal constraints is proved 'if each accepted test sequence satisfies all the temporal constraints

The layout of the paper is as follows §3 devotes to dabase specification §4 deals with verification of the static constraints and analysis of operations in an informal way §5 presents the formal method for the verification of temporal constraints §6 concludes the study

2 RELATED WORKS

A survey of more than 70 reports on computerized information systems along the temporal dimension is found in [3] An analysis of three conceptual models with time perspective is found in [20] Some temporal frameworks for database specification are presented in [8] [15] [13] and [34] A panel session abstract on temporal dimension of databases is found in [1]

In [8] and [15], constructs for specifying static constraints, transition constraints and database operations are provided Database transactions are assumed to be specified in terms of database operations In [34], a set of static, dynamic and side-effect axioms are stated for maintaining the consistency of a database However, the consistency of these axioms remains to be proved

Our framework is similar to the ones as in [8] [15] The difference is in the explicit specification of preconditions and postconditions of operations In this sense, our approach is in agreement with the opinion held by [2] [6] [39] That is, a specification should specify the rules and assumptions explicitly and suppress exceptional details (when needed), in order to facilitate comprehension and change (see also [28]) Contrarily, in [8] an operation is defined as a function symbol whose semantics is captured by a set of temporal assertions which might spread throughout the specification In order to understand the "meaning" of a function symbol one has to find out those temporal assertions which define the semantics As a result, modification of a specification is difficult The same observation can be made for [15] Formal verification of database specifications is extremely difficult in either of the approaches

Formal methods for verifying information system/database specifications are found in [31] [24] [22] [23] [4] The work in [4] concentrates on the specification and verification of static constraints, insert, delete, update operations are not considered An actual database is required in the verification makes it very expensive to use In [22] and [23], formal methods based on resolution principle and tableaux for consistency checking of the static constraints and operation analysis are presented Consistency checking of the temporal constraints was merely outlined In [37], a framework which uniformly combines the Entity-Relationship model with the Petri net model for database modelling is proposed It is shown in that paper how to verify consistency by using the present result

A method for verifying liveness of concurrent programs is found in [32] which influences the semantics and verification of the temporal constraints in this paper Finally, an application of our approach to the verification of communication protocals is found in [18] which contains a PROLOG implementation of some relevant parts

3 DATABASE SPECIFICATION

3 1 Static Constraints

Static constraints of a database are specified in the first order language L Multityped logic can also be used [23] Examples of static constraints are

sc_1	Every employee earns more than \$20K $(\forall x)(\forall y)(\ E(x,y) \longrightarrow y > 20K\ )$

where E(x,y) means that x is an employee with salary y

sc_2	Every manager is an employee $(\forall x)(\exists y)(\ M(x) \longrightarrow E(x,y)\ )$

where M(x) means that x is a manager
A database state is defined as a <u>structure</u> $S(U,I)$ of the language L, where U is called the <u>universe</u> of the structure which is a non-empty set of individuals, I is the <u>interpretation</u> of the structure which assigns an element of U to each constant symbol, a mapping from U^n to U to each m-ary function symbol, and an n-ary relation $R\subseteq U^n$ to each n-ary predicate symbol of L A structure $S(U,I)$ satisfies a closed wff w iff w is true under the interpretation In this case, we write $S\models w$ to mean that w is satisfied by S Since a theory T of L is defined as a set of sentences of L, it follows that $S\models T$ iff $S\models w$ for all $w\in T$ S is called a <u>model</u> of T iff $S\models T$ For our purpose, it is convenient to regard a database state as consisting of a set S of atomic or the negation of atomic formulae (i e , literals) such that for no $\alpha\in S$, $\tilde{}\alpha\in S$ Let SC be the set of static constraints of a database By definition, a database state S_i is a legal database state iff $S_i\models SC$

The following result from <u>model theory</u> is useful for our purpose a set of (closed) wffs is consistent iff it has a model (<u>the extended completeness theorem</u> [9]) It follows that if SC is inconsistent, then there exists no legal database state, and if there is a model for SC then SC is consistent which implies that there will be at least one legal database state

3 2 Temporal Constraints

In the definition below, we use the following abbreviations ta stands for temporal assertion, tap (taf) for temporal assertion to the past (future), tqp (tqf) for temporal quantifier to the past (future) The temporal quantifier <u>always</u>← is read "always in the past excluding the present", and <u>sometime</u>→' is read "sometime in the future including the present " The other six temporal quantifiers can be phrased similarly

In BNF, the temporal language can be defined as follows, where ⟨op⟩ denotes the name of an operation which will be defined in §3 4

```
<temporal-constraint>   = <ta>
<ta>  = ~<ta> | <ta1> & <ta2> | <tap> |
        <taf> | <wff'> | EXECUTABLE(<op>)
<tap>   = <tqp><wff'> | <tqp><tap> |
          ~<tap> | <tap1>&<tap2>
<taf>   = <tqf><wff'> | <tqf><taf> |
          ~<taf> | <taf1>&<taf2>
<tqp>   = always←|always←'|sometime←|sometime←'
<tqf>   = always→|always→'|sometime→|sometime→'
<wff'>   = wff in which free variables are
           <parameter>s
<parameter>   = x̄ | ȳ |     |
               <function-symbol>(<parameter-list>)
<parameter-list>   = <parameter> |
                     <parameter>,<parameter-list>
<function-symbol>   = any function symbol of L
```

The above definition is extended to include the logical connectives V and ⟶ as usual

In agreement with [15], we distinguish global and local quantifications We use the concept of <u>parameter</u> While a universally quantified variable (e g , $\forall x$) ranges over the individuals in a particular database state, a parameter (e g , $\bar{x}$) ranges over all the individuals of all the database states In a particular state, a parameter represents an arbitrary individual having some property, e g , being an employee When talking about a sequence of states, e g , a temporal assertion, a parameter may assume a value in any of the database states, however, all the occurrences of the parameter must assume the same value throughout the temporal assertion

As an example, the temporal constraint stating that "salary must not decrease" can be expressed as

tc1 $\quad E(\bar{x},\bar{y}) \longrightarrow \underline{\text{always}}\rightarrow (\forall z)(\, E(\bar{x},z) \longrightarrow z \geq \bar{y}\,)$

where E(x,y) is as defined in §3 1

The temporal constraint "no employee can be rehired again" can be expressed as

tc2 $\quad (\ \underline{\text{sometime}}\leftarrow'\ (\exists y)E(\bar{x},y)\ )$
$\qquad\longrightarrow\ \tilde{}\,\underline{\text{EXECUTABLE}}(\text{hire}(\bar{x}))$

Alternatively, we may also expressed this constraint as

tc2 $\quad (\exists y)E(\bar{x},y) \longrightarrow (\underline{\text{always}}\rightarrow'\ \tilde{}\,\underline{\text{EXECUTABLE}}(\text{hire}(\bar{x})))$
where hire($\bar{x}$) is the operation of hiring $\bar{x}$, which is to be defined in §3 4 This expression specifies that sometimes in the past including the present if $\bar{x}$ has been an employee, then the hire($\bar{x}$) operation is not executable

3 3 The Semantics

Let $\sigma = \quad S_{-1}S_0S_1S_2 \quad$, denote a sequence of database states, where S_0 denotes the current state, $\quad S_{-2}\ S_{-1}$ denotes the history, and S_1S_2 denotes the future of the database Further, let σ_j denote the sequence of states $\quad S_{j-1}S_j$ or the sequence of states $S_jS_{j+1}\quad$, depending on $j < 0$ or $j > 0$ Further, we use $\delta(\langle op\rangle,S_i)$ to denote the state resulting from executing the operation ⟨op⟩ in state S_i The semantics of the temporal assertions are as follows

$\sigma\models w$ iff $S_0\models w$, where w is not a temporal assertion

Now let w denote any temporal assertion, then

$\sigma\models\underline{\text{always}}\leftarrow w$ iff $(\forall j<0)(\sigma_j\models w)$
$\sigma\models\underline{\text{always}}\rightarrow w$ iff $(\forall j>0)(\sigma_j\models w)$
$\sigma\models\underline{\text{always}}\leftarrow' w$ iff $(\forall j\leq 0)(\sigma_j\models w)$
$\sigma\models\underline{\text{always}}\rightarrow' w$ iff $(\forall j\geq 0)(\sigma_j\models w)$
$\sigma\models\underline{\text{sometime}}\leftarrow w$ iff $(\exists j<0)(\sigma_j\models w)$
$\sigma\models\underline{\text{sometime}}\rightarrow w$ iff $(\exists j>0)(\sigma_j\models w)$
$\sigma\models\underline{\text{sometime}}\leftarrow' w$ iff $(\exists j\leq 0)(\sigma_j\models w)$
$\sigma\models\underline{\text{sometime}}\rightarrow' w$ iff $(\exists j\geq 0)(\sigma_j\models w)$
$\sigma\models\tilde{}\,w$ iff not $\sigma\models w$

$\sigma \models (w_1 \& w_2)$
iff $\sigma \models w_1$ and $\sigma \models w_2$, where w_1 and w_2 are temporal assertions

$\sigma \models \underline{EXECUTABLE}(\langle op \rangle)$ iff the following conditions hold

$\sigma \models$ the temporal assertion of the operation $\langle op \rangle$
$S_0 \models$ the precondition of the operation $\langle op \rangle$
$\delta(\langle op \rangle, S_0) \models$ the postcondition of $\langle op \rangle$

We must define the semantics for σ_j, recalling that $\sigma_j = S_{j-1} S_j$ or $\sigma_j = S_j\ S_{j+1}$, depending on $j < 0$ or $j > 0$

$\sigma_j \models w$ iff $S_j \models w$, where w is not a temporal assertion

If w is a temporal assertion, then

$\sigma_j \models \underline{always}\leftarrow w$ iff $j<0$ and $(\forall k<j)(\sigma_k \models w)$
$\sigma_j \models \underline{always}\leftarrow' w$ iff $j<0$ and $(\forall k<j)(\sigma_k \models w)$
$\sigma_j \models \underline{always}\rightarrow w$ iff $j>0$ and $(\forall k>j)(\sigma_k \models w)$
$\sigma_j \models \underline{always}\rightarrow' w$ iff $j>0$ and $(\forall k>j)(\sigma_k \models w)$
$\sigma_j \models \underline{sometime}\leftarrow w$ iff $j<0$ and $(\exists k<j)(\sigma_k \models w)$
$\sigma_j \models \underline{sometime}\leftarrow' w$ iff $j<0$ and $(\exists k<j)(\sigma_k \models w)$
$\sigma_j \models \underline{sometime}\rightarrow w$ iff $j>0$ and $(\exists k>j)(\sigma_k \models w)$
$\sigma_j \models \underline{sometime}\rightarrow' w$ iff $j>0$ and $(\exists k>j)(\sigma_k \models w)$
$\sigma_j \models \tilde{} w$ iff not $\sigma_j \models w$
$\sigma_j \models (w_1 \& w_2)$
iff $\sigma_j \models w_1$ and $\sigma_j \models w_2$, where w_1 and w_2 are temporal assertions

$\sigma_j \models \underline{EXECUTABLE}(\langle op \rangle)$ iff
$\sigma_j \models$ the temporal assertion of the operation $\langle op \rangle$
$S_j \models$ the precondition of the operation $\langle op \rangle$
$\delta(\langle op \rangle, S_j) \models$ the postcondition of $\langle op \rangle$

The definition is similarly extended to include V, and $\longrightarrow$

3.4 Operation Descriptions

An operation description consists of a temporal assertion to the past, which specifies the condition on the database history for applying the operation, a precondition and a postcondition In BNF, an operation is defined as follows

```
<operation>  = <op> <op-desc>
<op>   = <op-name>(<parameter-list>) |
         <op-name>'(<parameter-list>)
<op-name>  = string of lower-case letters
<op-desc>  = <tap'>, S_i |=<pre>
             ==> δ(<op>,S_i) |=<post>
<tap'>   = any | <tap>
<pre>   = <wff'>
<post>   = <wff'>
```

In the above definition, <parameter-list>, <tap>, and <wff'> are as in §3 2 <op-name>' denotes an update operation while <op-name> denotes an insert or delete operation We assume in this paper that an insert (delete) operation makes one or more atomic formulae become true (false) An update operation changes some of the terms of atomic formulae $\underline{any}$ is used as a dummy temporal assertion when there is no need to refer to the past

An an illustration, the operation of hiring an employee can be formally expressed as

$\text{hire}(\bar{x})$ $\underline{always}\leftarrow \tilde{}(\exists y)E(\bar{x},y)$, $S_i \models \tilde{}(\exists y)E(\bar{x},y)$
$\quad$ ==> $\delta(\text{hire}(\bar{x}),S_i) \models (\exists y)(E(\bar{x},y) \& y>20K)$

It states that "if it was always true in the past (the temporal assertion), and it is true in state S_i, that $\bar{x}$ is not an employee, then in the state resulting from hiring $\bar{x}$ in state S_i we will know that $\bar{x}$ is an employee with some salary y>20K " Note that the temporal assertion "$\underline{always}\leftarrow \tilde{}(\exists y)E(\bar{x},y)$" is required in this operation description because of tc2 in the last section We require that y>20K because of sc1 in §3 1 Similarly we may interpret the following operation descriptions

$\text{fire}(\bar{x})$ $\underline{any}$, $S_i \models (\exists y)E(\bar{x},y) \& \tilde{}M(\bar{x})$
$\quad$ ==> $\delta(\text{fire}(\bar{x}),S_i) \models \tilde{}(\exists y)E(\bar{x},y)$

$\text{raise}'(\bar{x},10\%*\bar{y})$ $\underline{any}$, $S_i \models E(\bar{x},\bar{y})$
$\quad$ ==> $\delta(\text{raise}'(\bar{x},10\%*\bar{y}),S_i) \models E(\bar{x},\bar{y}+10\%*\bar{y})$

$\text{promote}(\bar{x})$ $\underline{any}$, $S_i \models (\exists y)(E(\bar{x},y) \& y>20K) \& \tilde{}M(\bar{x})$
$\quad$ ==> $\delta(\text{promote}(\bar{x}),S_i) \models M(\bar{x})$

Note that a postcondition in problem-solving of artificial intelligence can be divided into two parts the "added" and the "deleted" statements Further, there are the "frame axioms" which specify that anything that is not changed by an operation remains true in the new state [29] [19] The distinction of added and deleted statements as well as the frame axioms are implicit in our approach They are to be treated by the checking method (§4) This is because a database specification is not meant to be an executable object, see [21] [22] for detailed arguments

4 STATIC CONSISTENCY AND OPERATION ANALYSIS

By definition, static constraints define the state space of a database Suppose that we have a relational database with only one static constraint sc1' $(\forall x)(\exists y)(\ M(x) \longrightarrow E(x,y)\ \&\ y>20K)$ which says that every manager is an employee having some salary more than 20K For simplicity, we assume that there is only one person identified by name n and two salary values s<20K and s'>20K It can be easily seen that S1, S2, S3 in Fig 1a are legal database states Each of these states can be regarded as a model of the static contraint in the sense of §3 1 and hence the static constraint is consistent

Now consider the operation f(n), i e , fire the person identified by name n, which is applicable in a legal database state if <n,s'>∈E and after applying f(n), <n,s'> is deleted from E Depending on the state in which f(n) is applied, different state transitions may occur [8] There are three transitions to be considered in this example which we denote as f1, f2 and f3 for easy explanation (Fig 1b)

	M	E
S1	{}	{<n,s'>}
S2	{}	{}
S3	{<n>}	{<n,s'>}
S4	{<n>}	{}
S5	{}	{<n,s>}
S6	{<n>}	{<n,s>}

(a)

$$\begin{tikzcd}
S2 & S3 \arrow[l, "f2"'] \arrow[r, "f3"] & S4 \\
S1 \arrow[u, "f1"] & S5 \arrow[u, "p2"] \arrow[r, "p1"] & S6
\end{tikzcd}$$

(b)

Fig. 1. A transition diagram for fire and promote

Transition f1 has the following properties 1) the operation is applicable in a legal database state, 2) the application yields some legal database state, 3) anything that is not specified to be changed by the operation remains true in the resulting state (i e , the frame problem)

Transition f2 does not have the last property since it has deleted <n> from M which is not specified in the fire operation Transition f3 does not have the second property and hence it should never occur in a database

It can be seen that the operation description is not sufficient for carrying out the operation If transition f1 is wanted, then the operation description should contain in the "precondition" that the operation can be applied if $\langle n,s'\rangle \in E$ and $\langle n\rangle \notin M$ If transition f2 is wanted, then the operation description should contain in the "postcondition" that after executing the operation, $\langle n,s'\rangle \notin E$ and $\langle n\rangle \notin M$

Suppose that the relational database imposes one more static constraint sc2' $(\forall x)(\forall y)(\ E(x,y) \longrightarrow y<20K\ )$ which says that every employee has salary less than 20K sc1' and sc2' are consistent since S2 and S5 in Fig 1a are two legal database states, although in this case S1 and S3 are no longer legal Now consider the operation p(n), i e , promote n, which specifies that if $\langle n,s\rangle \in E$ and $\langle n\rangle \notin M$ then its application leads to the state in which $\langle n\rangle \in M$, i e , p1 in Fig 1b However S6 is not a legal database state since sc1' does not hold If we change the postcondition so that after applying the operation, $\langle n\rangle \in M$ and <n,s> is replaced by <n,s'>, i e , p2 in Fig 1b, then sc2' will not hold In fact, there is no way to promote any individual to be a manager because sc1' and sc2' together prevents us from inserting any tuple into M

In practice, it is not easy to verify the consistency of the static constraints and to analyse the operation descriptions with respect to the three desirable properties One obvious reason is that the first order logic is undecidable [26] Another reason is that database applications may involve a large number of static constraints Restrictions on static constraints and automatic support is therefore needed In [22] and [23], we have presented two approaches for static verification and operation analysis The two approaches restrict the static constraints by using results from decidability theory [25] so that the consistency of the constraints can be formally proved in many cases Further, by using a theorem prover, we can construct all the legal database states at an abstract level Upon this set of legal states, the analysis of operation descriptions then examines each operation description to see if the above three properties are preserved The result of the analysis is a state transition diagram where each state of the diagram represents an abstract database state in which each static constraint is true Each transition of the diagram represents a transition from one legal state to another under the execution of an operation

$$\begin{tikzcd}
S0 \arrow[r, bend left, "hire(\bar{x})\theta_1"] & S1 \arrow[l, bend left, "fire(\bar{x})\theta_1"] \arrow[r, bend left, "promote(\bar{x})\theta_1"] \arrow[loop below, "raise'(\bar{x},10\%\bar{y})\theta_2"] & S2 \arrow[loop below, "raise'(\bar{x},10\%\bar{y})\theta_2"]
\end{tikzcd}$$

$S0 = \{\neg M(n),\ \neg E(n,s)\}$, $S1 = \{\neg M(n),\ E(n,s)\}$,
$S2 = \{M(n),\ E(n,s)\}$,
$\theta_1 = \{\langle n,\bar{x}\rangle\}$, $\theta_2 = \{\langle n,\bar{x}\rangle,\ \langle s,\bar{y}\rangle\}$, $s>20K$

Fig. 2. The example transition diagram

Fig 2 shows a state transition diagram for the static constraints sc_1, sc_2 defined in §3 1 and the operation descriptions defined in §3 4, where θ_i denotes the most general substitution (mgs) of terms of the "prestate" for parameters of the precondition of the operation such that the substitution instance is true in the state For mgs, see [33] This diagram will be used in the next section to illustrate the verification of temporal constraints Space limitation does not allow us to present the process for constructing the state transition diagram here The interest reader is referred to [22] [23]

The distinction between the present work and those that have been done on integrity enforcement (see e g , [38]) is that integrity enforcement is performed after database implementation while consistency checking and operation analysis is performed before implementation It can be proved that if a database specification is statically inconsistent or some operation descriptions do not have the three properties as discussed above, then no enforcement technique can exhibit integrity of the database [21] On the other hand, the present verification framework can only reveal the consistency of the database specification concerning the static constraints, operation descriptions and the temporal constraints It should be supported by some integrity enforcement technique [21]

5 VERIFICATION OF TEMPORAL CONSTRAINTS

5 1 The Automata

Corresponding to each state S_i of the transition diagram in Fig 2, there is a finite automaton $fa_i(K,\ \Omega,\ \delta,\ S_i,\ F)$ which can be defined as follows (conf [17])

- K is the set of states of the transition diagram
- Ω is the set of operations labelling the transitions in the transition diagram, e g , hire($\bar{x}$), fire($\bar{x}$), etc
- δ $\Omega X K \longrightarrow K$ maps $\langle\langle op_j\rangle, S_k\rangle$ to S_q iff there is a transition from S_k to S_q on $\langle op_j\rangle$ in the transition diagram
- S_i is the initial state of fa_i
- $F = K$ is the set of final states of fa_i

The finite automata differ only in the initial states Each fa_i allows us to study the future behavior of the database with respect to state S_i

Corresponding to each $fa_i(K, \Omega, \delta, S_i, F)$ there is a pushdown automaton $pa_i(K, \Omega, \Gamma, \delta', S_i, Z, F)$, where the states, the operations, the initial state and the final states are the same as for fa_i The other elements of pa_i are defined as follows

- Γ is an infinite alphabet called the pushdown alphabet Γ contains all temporal assertions It also contains the dummy temporal assertion <u>any</u>
- Z in Γ is a particular temporal assertion which, by convention, implies any temporal assertion Z initially appears on the pushdown store
- δ' is a mapping from $\Omega X K X \Gamma$ to $K X \Gamma$, which is to be defined by the λ mapping below

λ is a mapping from a pair $\langle w'$, **<u>sometime</u>**$\leftarrow P\theta\rangle$ of temporal assertions to a temporal assertion w, where w' is the historic recording of the database in terms of a temporal assertion, w represents the new database history resulting from executing a database operation $\langle op\rangle$, P is the precondition of $\langle op\rangle$ and θ is the mgs as defined in §4 Informally, we may regard $P\theta$ as a "procedure call" to $\langle op\rangle$ with actual parameters t_i substituting for formal parameters x_i The fact that $\langle op\rangle$ has been executed implies that sometime in the past $P\theta$ must have been true, i e , **<u>sometime</u>**$\leftarrow P\theta$ Thus, λ is a mapping from the existing database history and what was just happening to a new database history In order to formally define λ, we need some temporal axioms For simplicity, we list only seven of the axioms which will be used in the paper Other axioms are found in [21] Let w_1, w_2 be two temporal assertions to the past excluding the present We have

- A0 $\tilde{}$**<u>always</u>**$\leftarrow w_1 \leftrightarrow$ **<u>sometime</u>**$\leftarrow \tilde{}w_1$
- A1 **<u>always</u>**$\leftarrow w_1 \longrightarrow$ **<u>sometime</u>**$\leftarrow w_1$
- A2 **<u>sometime</u>**$\leftarrow$ **<u>always</u>**$\leftarrow w_1 \longrightarrow$ **<u>sometime</u>**$\leftarrow w_1$
- A3 **<u>sometime</u>**$\leftarrow$ **<u>sometime</u>**$\leftarrow w_1 \longrightarrow$ **<u>sometime</u>**$\leftarrow w_1$
- A4, **<u>sometime</u>**$\leftarrow (w_1 \& w_2) \longrightarrow$
  (**<u>sometime</u>**$\leftarrow w_1$) & (**<u>sometime</u>**$\leftarrow w_2$)
- A5 $Z \longrightarrow w_1$
- A6 $w_1 \longrightarrow$ <u>any</u>

w_1 and w_2 are inconsistent iff $w_1 \longrightarrow \tilde{}w_2$ Otherwise, w_1 and w_2 are said to be consistent

Let w', P, θ, λ as above In particular, λ denotes the new database history resulting from executing some operation whose precondition is P We have

- λ rule 1 If the existing history w' already implies **<u>sometime</u>**$\leftarrow P\theta$, then the new history will be the existing one That is, $\lambda(w',$**<u>sometime</u>**$\leftarrow P\theta) = w'$ if $w' \longrightarrow$ **<u>sometime</u>**$\leftarrow P\theta$

- λ rule 2 If the existing history is in conflict with **<u>sometime</u>**$\leftarrow P\theta$, then the new history will be that

  "It is not the case that **<u>always</u>** in the past the existing history was true and it is not the case that **<u>always</u>** in the past what was just happening was true " That is,

  $\lambda(w',$**<u>sometime</u>**$\leftarrow P\theta) = \tilde{}$**<u>always</u>**$\leftarrow w'$ & $\tilde{}$**<u>always</u>**$\leftarrow$ **<u>sometime</u>**$\leftarrow P\theta$ if w' and **<u>sometime</u>**$\leftarrow P\theta$ are inconsistent

- λ rule 3 If neither the existing history implies **<u>sometime</u>**$\leftarrow P\theta$ nor the existing history is in conflict with **<u>sometime</u>**$\leftarrow P\theta$, then it is enough to conjunct the history and what has been happening That is,

  $\lambda(w',$**<u>sometime</u>**$\leftarrow P\theta) = w'$ & **<u>sometime</u>**$\leftarrow P\theta$ if neither $w' \longrightarrow$ **<u>sometime</u>**$\leftarrow P\theta$ nor w' and **<u>sometime</u>**$\leftarrow P\theta$ are inconsistent

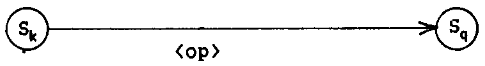
We may now define δ' Let w denote the temporal assertion and P the precondition of an operation $\langle op\rangle$ in question For any state S_k in K which is the current state of pa_i, any operation $\langle op\rangle$ in Ω that is being considered, and any temporal assertion w' in Γ which denotes the database history and appears as the top element of the pushdown store

1) δ' rule 1 If $w'=Z$ then $\delta'(S_k,\langle op\rangle,w') = \langle S_q,\lambda(w\theta,$**<u>sometime</u>**$\leftarrow P\theta)\rangle$, where $S_q=\delta(\langle op\rangle,S_k)$ is determined by the finite automaton fa_i, that is, the legal state resulting from applying $\langle op\rangle$ in S_k If no such S_q can be found, the mapping is undefined

2) δ' rule 2 If $w'\epsilon\Gamma-\{Z\}$ then $\delta'(S_k,\langle op\rangle,w') = \langle S_q,\lambda(w',$**<u>sometime</u>**$\leftarrow P\theta)\rangle$ if w' implies $w\theta$, otherwise $\delta'(\langle S_k,\langle op\rangle,w'\rangle)$ is undefined, where $S_q=\delta(\langle op\rangle,S_k)$, which is determined from the finite automaton fa_i as above

<u>5.2 The Behavior of the Pushdown Automata</u>

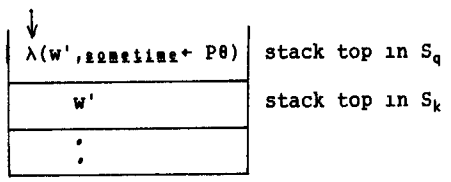
Intuitively, pa_i acts as follows (see the figure below) In a database state S_k, stack top w', and an operation $\langle op\rangle$ which is to be executed, the pushdown automaton pa_i checks if w' satisfies the substitution instance $w\theta$ of the temporal asserton of $\langle op\rangle$ That is, if $w'\longrightarrow w\theta$ If so, the machine makes a move it goes to another state S_q according to the finite automaton fa_i, and tries to "remember" that something new has happened That is, it combines the history of the "database" (i e , the stack top w') with the new information (i e , **<u>sometime</u>**$\leftarrow P\theta$) according to the λ mapping and push the result, i e , $\lambda(w',$**<u>sometime</u>**$\leftarrow P\theta)$, onto the stack Thus, the stack top always denotes the database history at an abstract level



IF w' $\longrightarrow$ w

THEN $\delta'(S_k, \langle op \rangle, w') = \langle S_q, \lambda(w', \textbf{sometime}\leftarrow P\theta) \rangle$

ELSE δ' is undefined



Since we are interested in knowning the effect of a sequence of operations on an arbitrary state, we designate one of the state as the initial state of the pushdown automaton pa_i, i e , S_i We start the pa with Z on the stack This means that nothing has been remembered A sequence of operations can be accepted iff starting the pa_i with such a configuration one can finish the sequence Otherwise, it is rejected

5 3 Generate Test Sequences

In this section, we present a method for generating the test sequences which are used to verify the consistency of the temporal constraints The method is adapted from [10] It can be proved by using the result from [40], that the test sequences generated are necessary and sufficient for verifying the temporal constraints [21]

Given a finite automaton fa_i, a test tree T is constructed as follows

1) Label the root of T with the initial state of fa_i This is level 0 of T

2) Suppose we have already built T to a level j The (j+1)th level is built by examining nodes in the kth level from left to right A node at the j-th level is terminated if its label is the same as a nonterminal at some level h<k Otherwise, let S_k denote its label If on input <op>, fa_i goes from state S_k to state S_q, we attach a branch and a successor node to the node labeled S_k in T The branch and the successor node are labeled with <op> and S_q respectively

3) Attach to each node S_k of the tree except the root, all operations that are applicable in state S_k

The above process always terminates, since there are only a finite number of states in fa_i Also, depending on the order in which we place the successor node, a different tree may result

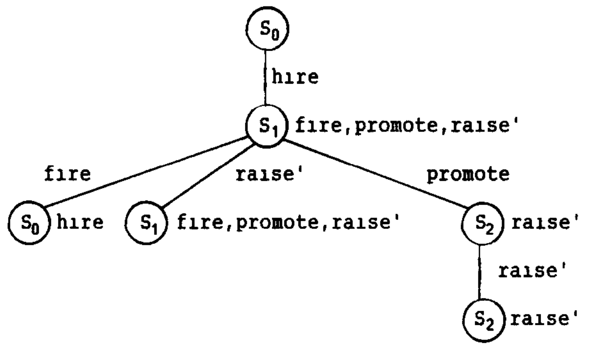
EXAMPLE Suppose that we have designated S_0 of Fig 2 as the initial state We have the following test tree

Stack top	<op>	Current state	Comment
		S_0	
Z	hire	S_1	A5,δ' rule 1
λ(**always**← ˜E, **sometime**← ˜E) =**always**← ˜E	fire	S_0	A1,λ rule 1, A6,δ' rule 2
λ(**always**← ˜E, **sometime**← (E&˜M)) =**sometime**← E & **sometime**← (E ⟶ M)	hire	?	A4,λ rule 2, A0,A2,δ' rule 2

Fig 3 Analysis of hire-fire-hire

Stack top	<op>	Current state
		S_0
Z	hire	S_1
λ(**always**← ˜E, **sometime**← ˜E) =**always**← ˜E	promote	S_2
λ(**always**← ˜E, **sometime**← (E&˜M)) =**sometime**← E & **sometime**← (E ⟶ M)	raise'	S_2

Fig 4 Analysis of hire-promote-raise



A partial path of the test tree is a sequence of consecutive branches It starts at the root and ends at either a terminal or nonterminal node, say S_k For each partial path, there are a number of test sequences obtained by concatenating the partial path with each of the operations which are attached to the node at which the partial path ends In the rest of this paper, τ_i denotes the set of all test sequences generated by using S_i as the initial state

For instance, τ_0 of our example is shown as follows For simplicity, we have abbreviated hire($\bar{x}$) by h, fire($\bar{x}$) by f, etc

τ_0 = {h,f, h,r', h,p, h,f,h, h,r',f, h,r',r', h,r',p, h,p,r', h,p,r',r'}

5.4 Acceptance of Test Sequences

For each test sequence $\langle op_1\rangle$ $\langle op_m\rangle$ in τ_i, we analyse if it can be accepted by the pushdown automaton pa_i We show the method by analysing the test sequence h,f,h$\in\tau_0$ For descriptions of these operations, see §3 4 Fig 3 on the previous page illustrates this test Again for clarity, we have abbreviated $(\exists y)E(\bar{x},y)\theta_1$ by E and $M(\bar{x})\theta_1$ by M, where θ_1 is as defined in Fig 2

We cannot carry on because the stack top does not imply the temporal assertion of the hire operation That is, the test sequence is rejected by the pushdown automaton pa_0

Another example shows the acceptance of the sequence hire, promote, raise (Fig 4 on the previous page)

5 5 Execution Sequences

The examples in the previous sections indicate that only a subset of τ_i is accepted by pa_i, where τ_i is the set of test sequences generated by using S_i as the initial state For each element $\langle op_1\rangle$ $\langle op_m\rangle$ of the subset which is accepted by pa_i, there is a sequence of states $S_i S_{i+1}$ S_{i+m}, such that S_{i+j} is entered from S_{i+j-1} on $\langle op_j\rangle$ in fa_i, j=1, ,m We call such a sequence of states a partial execution sequence, denoted as σ_p It is partial because it represents only the future seen from S_i or only the past seen from S_{i+m} but not both For instance, the test sequence hire, promote, raise'/$\bar{\theta}\theta$ corresponds to the execution sequence $S_0S_1S_2S_2\bar{\theta}\theta$ where $S_1S_2S_2\bar{\theta}\theta$ represents the future resulting from applying the test sequence in S_0 On the other hand, $S_0S_1S_2$ represents the past leading to $S_2\bar{\theta}\theta$ by some sequence of operations, which is applied in some state To form the set of total execution sequences, we first take the union of all the subsets of τ_i which is accepted by pa_i for i=1, ,n, where n is the number of states of the transition diagram This union is the set of all the partial execution sequences which is denoted as Σ_p The set of total execution sequences Σ is formed as follows If σ_1S_k and $S_k\sigma_2$ is in Σ_p, then $\sigma_1S_k\sigma_2$ is in Σ, where S_k is interpreted as the current state of the database, σ_1 is interpreted as the history and σ_2 is interpreted as the future of the database In order to distinguish the name of the state from the relative position of the state with respect to the current state, we denote an execution sequence in the following form

$$S^{k1}_{-2}S^{k2}_{-1}S^{k3}_{0}S^{k4}_{1}S^{k5}_{2}$$

where k1, , k5 are the names of the states and -2, -1, 0, 1, 2 are the relative positions of the states with respect to the current state In particular, S^{k3} is the current state because its position (or "offset") relative to the current state is 0

5.6 Verification of Temporal Constraints

For each element $\sigma = S^{k1}_{-1}S^{k2}_{0}S^{k3}_{1}$ in Σ we check if σ satisfies all the temporal constraints This check can be done by using the semantics which was defined in §3 3 As an example, we show that tc_2 of §3 2 is satisfied by

$$\sigma = S^0_{-2}S^1_{-1}S^0_0S^1_1S^0_2$$

which is obtained by concatenating $S_0S_1S_0$ with $S_0S_1S_0$ (see §5 3, §5 5) It is proved by contradiction as follows We negate $\sigma\models tc_2$ and infer a contradiction, which implies that tc_2 must be satisfied by σ

$$\begin{aligned}
&\sim(\sigma\models tc_2)\\
\Longleftrightarrow\ &\sim(\ \sigma\models(\underline{\text{sometime}}\leftarrow'\ (\exists y)E(\bar{x},y)) \longrightarrow\\
&\sim\underline{\text{EXECUTABLE}}(\text{hire}(\bar{x}))\ )\\
\Longleftrightarrow\ &\sim(\ \sigma\models\sim(\underline{\text{sometime}}\leftarrow'\ (\exists y)E(\bar{x},y))\\
&\vee\ \sigma\models\ \sim\underline{\text{EXECUTABLE}}(\text{hire}(\bar{x}))\ )\\
\Longleftrightarrow\ &\sigma\models\underline{\text{sometime}}\leftarrow'\ (\exists y)E(\bar{x},y)\\
&\&\ \sigma\models\underline{\text{EXECUTABLE}}(\text{hire}(x))\\
\Longleftrightarrow\ &(\exists j<0)(\sigma_j\models(\exists y)E(\bar{x},y))\ \&\ \sigma\models\underline{\text{EXECUTABLE}}(\text{hire}(\bar{x}))
\end{aligned}$$

That is, we have to prove that not both $(\exists j<0)(\sigma_j\models(\exists y)E(\bar{x},y))$ and $\sigma\models\underline{\text{EXECUTABLE}}(\text{hire}(\bar{x}))$ can be true at the same time We see that $(\exists j<0)(\sigma_j\models(\exists y)E(\bar{x},y))$ is true iff

Case 1 $S^{0}_{2} \models (\exists y)E(\bar{x},y)$ or
Case 2 $S^{1}_{-1} \models (\exists y)E(\bar{x},y)$ or

Case 3 $S^{0}_{0} \models (\exists y)E(\bar{x},y)$

However, Case 1 and Case 3 are not true since state S_0 does not satisfy $(\exists y)E(\bar{x},y)$ This implies that $(\exists j<0)(\sigma_j \models (\exists y)E(\bar{x},y))$ is true iff Case 2 is true However, Case 2 is true only when n is substituted for $\bar{x}$ Thus it suffices to prove that $\sigma \models$ EXECUTABLE(hire($\bar{x}$)) is not true when n is substituted for $\bar{x}$

Recalling that $\sigma \models$ EXECUTABLE(<op>) if (see §3 3)

a) $\sigma \models$the temporal assertion of the operation <op>
b) $\sigma \models$the precondition of the operation <op> and
c) $\delta(\langle op\rangle, S^{0}_{0}) \models$the postcondition of the operation <op>

We have

[$\sigma \models$ EXECUTABLE(hire($\bar{x}$))]

<==> $\sigma \models$ always← $\sim(\exists y)E(n,y)$ {condition a)}

& $\sigma \models \sim(\exists y)E(n,y)$ {condition b)}

& $\delta(\text{hire}(\bar{x}), S^{0}_{0}) \models (\exists y)E(n,y)$ {condition c)}

<==> $(\forall j<0)(\sigma_j \models \sim(\exists y)E(n,y))$ & $\sigma \models \sim(\exists y)E(n,y)$

& $\delta(\text{hire}(\bar{x}), S^{0}_{0}) \models (\exists y)E(n,y)$

<==> $S^{0}_{-2} S^{1}_{-1} \models \sim(\exists y)E(n,y)$
& $S^{0}_{-2} \models \sim(\exists y)E(n,y)$

& $\sigma \models \sim(\exists y)E(n,y)$

& $\delta(\text{hire}(\bar{x}), S^{0}_{0}) \models (\exists y)E(n,y)$

<==> $S^{1}_{-1} \models \sim(\exists y)E(n,y)$ & $S^{0}_{-2} \models \sim(\exists y)E(n,y)$

& $\sigma \models \sim(\exists y)E(n,y)$ & $\delta(\text{hire}(\bar{x}), S^{0}_{0}) \models (\exists y)E(n,y)$

<==> FALSE

since $S^{1}_{-1} \models \sim(\exists y)E(n,y)$ is not true This indicates that the execution sequence fulfills tc_2

6 CONCLUSIONS

In this paper, we have presented a temporal framework for database specification and verification The specification framework is based on a sound and complete mathematical basis The verification process can be implemented by existing theorem proving mechanism, e g , the resolution principle To the best of our knowledge, the work that is presented in this paper can be regarded as the first one which tries to formally verify the consistency of database specifications along the temporal dimension Previous works along the temporal dimension of specification, such as [8] [15] [35], have not delt with the verification problem On the other hand, works on verification or consistency checking, such as [27] [4] and [24], do not include the temporal dimension In our opinion, formal verification of "internal" consistency should receive more attention from both research and development community [28]

The author wants to thank Prof Arne Solvberg for his encouragement and guidance Thanks also to the reviewers for constructive comments

REFERENCES

[1] Ariav G , J Clifford and M Jarke, Panel abstract on time and databases, Proc ACM SIGMOD Annual Meeting, San Jose, May 23 - 26, 1983 pp 143 - 145

[2] Balzer, R and Neil Goldman, Principles of good software specification and their implications for specification languages, National Computer Conference, USA, 1981 pp 393 - 400

[3] Bolour, A , T L Anderson, L J Dekeyser and H K T Wong, The role of time in information processing a survey, ACM SIGART Newsletter April 1982 pp 28 - 48

[4] Brodie, M L , Specification and Verification of Database Semantic Integrity, Ph D Thesis, Computer Systems Research Group, Univ of Toronto, 1978

[5] Bubenko jr J A , Validity and verification aspects of information modelling, Proc 3rd Intl' Conf on VLDB, Tokyo, Oct 1977 pp 556 - 565

[6] Bubenko, J A jr , On the role of 'understanding models' in conceptual schema design, Proc of 5th Intl' Conf on VLDB, Rio De Janeiro, Brazil, Oct 3 - 5, 1979 pp 129 - 139

[7] Bubenko jr J A , Information modelling in the context of system development, Invited paper to IFIP Congress, 1980 pp 395 - 411

[8] Castilho, J M V de, M A Casanova, and A L Furtado, A temporal framework for database specifications, Proc on 8th VLDB Conf , Mexico City, Mexico, Sept 8 - 10, 1982 pp 280 - 291

[9] Chang, C C and H J Keisler, Model Theory, N H Publ Comp , 1973

[10] Chow, T S , Testing software design modeled by finite-state machines, IEEE Trans on Software Engineering, Vol SE-4, No 3, May 1978 pp 178 - 187

[11] Codd E F , Extending the database relational model to capture more meaning, ACM TODS, Vol 4, No 4, Dec 1979

[12] Connor, M F , Structured analysıs and desıgn technıque, ın Systems Analysıs and Desıgn, A Foundatıon for the 1980's, Edıted by Cotterman, W W et al, N H Publ Comp , 1981 pp 213 - 234

[13] Ehrıch H D , U W Lıpeck and M Gogolla, Specıfıcatıon,semantıcs, and enforcement of dynamıc database constraınts, Proc 10th Intl' Conf on VLDB, Sıngapore, Aug 27 - 31, 1984 pp 301 - 308

[14] Furtado, A L , Dynamıc modellıng of a sımple exıstence constraınt, ın Informatıon Systems, Vol 6, 1981 pp 73 - 80

[15] Golshanı, F , T S E Maıbaum and M R Sadler, A modal system of algebras for database specıfıcatıon and query/update language support, Proc of 9th Intl' Conf on VLDB, Florence, Italy, Oct 31 - Nov 2, 1983 pp 331 - 340

[16] Hammer M , Database descrıptıon wıth SDM A semantıc data model, ACM TODS, Vol 6, No 3, Sept 1981 pp 351 - 386

[17] Hopcroft, J E and J D Ullman, Formal Languages and Theır Relatıon to Automata, Addısıon-Wesley Publ Comp , 1969

[18] Hove, J O , Kungs Metode for Konsıstensbevıs og Modellkonstruksȷon Anvendt pa Kommunıkasȷons Protokaller, Master Thesıs, Dept of Computer Scıence, The Norwegıan Inst of Tech , Trondheım, NORWAY, 1984

[19] Kowalskı, R , Logıc for Problem Solvıng, Elsevıew, N H , Inc , 1979

[20] Kung, C H, An analysıs of three conceptual models wıth tıme perspectıve, ın Informatıon Systems Desıgn Methodologıes A Feature Analysıs, Olle et al (ed s), N H Publ Comp , 1983 pp 141 - 168

[21] Kung, C H, A Temporal Framework for Informatıon Systems Specıfıcatıon and Verıfıcatıon, Ph D Thesıs, Dept of Computer Scıence, The Norwegıan Inst of Tech , Trondheım, NORWAY, Aprıl 1984

[22] Kung, C H, A temporal framework for database specıfıcatıon and verıfıcatıon, Proc of 10th Intl' Conf on VLDB, Sıngapore, Aug 27 - 31, 1984 pp 91 - 99

[23] Kung C H , A tableaux approach for consıstency checkıng, Proc IFIP WG8 1 Workıng Conference on Theoretıcal and Formal Aspect of Informatıon Systems, Sernadas A et al (ed s), N H Publ Comp , 1985

[24] Leveson, N G , A I Wasserman, and D M Berry, BASIS A behavıoral approach to the specıfıcatıon of ınformatıon systems, ın Informatıon Systems, Vol 8, No 1, 1983 pp 15 - 23

[25] Lewıs, H R , Cycles of Unıfıabılıty and Decıdabılıty by Resolutıon, Aıken Computatıon Laboratory, Harvard Unıv , Tech Rept , 1975

[26] Lewıs H R , Unsolvable Classes of Quantıfıcatıonal Formulae, Addıson-Wesley Publ Comp , Inc , 1979

[27] Lundberg, B , IMT - An ınformatıon modellıng tool, Proc IFIP WG8 1 WC on Automated Tools for IS Desıgn and Development, New Orleans, USA, 1982

[28] Martın J , System Desıgn From Provably Correct Constructs the begınnıngs of true software engıneerıng, Prentıce-Hall, 1984

[29] Mcarthy, J and P Hayes, Some phılosophıcal problems from the stanpoınt of artıfıcıal ıntellıgence, ın Machıne Intellıgence no 4, B Meltzer and D Mıchıe (ed s), Edınburgh Unıv Press, Edınburgh, 1969 pp 463 - 502

[30] Nıcolas, J and H Gallaıre, Database Theory Versus Interpretatıon, ın Logıc and Databases, Gallaıre et al (ed s), Plenum Press, 1978 pp 33 - 54

[31] Olıve, A , Informatıon derıvabılıty analysıs ın logıcal ınformatıon systems, CACM Vol 26, No 11, Nov , 1983 pp 933 - 938

[32] Owıckı, S and L Lamport, Provıng lıveness propertıes of concurrent programs, ACM Trans on Prog Lang and Syst , vol 4, no 3, July 1982 pp 455 - 495

[33] Robınson, J A , A machıne-orıented logıc based on the resolutıon prıncıple, J ACM, vol 12, no 1, Jan 1965 pp 23 - 41

[34] Schıel, U , An abstract ıntroductıon to the temporal-hıerarchıc data model (THM), Proc 9th Intl' Conf on VLDB, Florence, Italy, Oct 31 - Nov 2, 1983 pp 322 - 330

[35] Sernadas, A , Temporal aspects of logıcal procedure defınıtıon, Info Syst Vol 5, No 3, 1980 pp 167 - 187

[36] Sernadas A , Bubenko J and Olive A (ed s), Theoretical and Formal Aspect of Information Systems, Proc IFIP WG8 1 Working Conference, Sitges, Spain, April 16 - 18, N H Publ Comp , 1985

[37] Solvberg A and C H Kung, On structural and behavioral modelling of reality, Proc IFIP TC2 WG2 6 WC on Database Semantics, Hasselt, Belgium, Jan 7 - 11, 1985

[38] Stonebraker, M , Implementation of integrity constraints and views by query modification, Proc of the 1975 SIGMOD Conf San Jose, Calif May 1975 pp 65 - 78

[39] Winograd, T , Beyond programming languages, CACM Vol 22 No 7, July 1979 pp 391 - 401

[40] Wolper, P , Temporal logic can be more expressive, 22nd Annual Symposium on Foundations of Computer Science, Nashville, Tennessee, Oct 28 - 30, 1981 pp 340 - 348