# A TEMPORAL FRAMEWORK FOR DATABASE SPECIFICATION AND VERIFICATION +

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 with respect to the static constraints 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, Temporal Database, Semantic Integrity, Formal Verification of Specifications.

## 1 INTRODUCTION

The data models and database description languages that have been proposed during the last decade can be roughly classified into four types:

1) Static models provide facilities for describing only a snapshot of the application. They may include process models which can be interpreted as sequences of computer instructions. Examples are the relational model of data, the Entity-Relationship model and the NIAM model.

2) Dynamic models provide declarative facilities for modelling state transitions without considering in full detail the mechanisms which achieve them. Usually a database operation is specified by a pair of one precondition and one postcondition. Examples are condition/event Petri-nets, ACM/PCM, BASIS [20], and [12].

3) Temporal models allow the specification of time dependent constraints such as "age must not decrease". Examples are found in [8][14][34]. Temporal logic is commonly used. Our approach is a combination of the dynamic and temporal approaches.

4) Full time perspective models emphasize the important role and particular treatment of time in modelling reality (see e.g., [7]). CIAM [6] belongs to this type. Other researches are found in [4][23][1].

Static approaches were proposed and focused by the mid-1970's, dynamic approaches started during the late 1970's and temporal/full time perspective approaches begin in the 1980's. More about the 4 types of models are in [19].

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. 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. 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 [26]. Second, it is shown in [11] and [7], that faults which are introduced during database specification and design have a major impact on systems development effort. 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) Specification of static constraints (§3.1). 2) Specification of temporal constraints (§3.3) and 3) Specification of operations (§3.5).

+ This research is financially supported by NORAD, NORwegian Agency for international Development.
* On leave from the Chinese Academy of Sciences, Beijing, China.

Correspondingly, the consistency checking method also consists of three parts: 1) Consistency checking of static constraints (§4.1). 2) Operation analysis (§4.2) and consistency checking of temporal constraints (§4.3) which will only be outlined in this paper. The interest reader is referred to [19].

## 2 RELATED WORKS

A survey of more than 70 reports about computerized information systems along the temporal dimension is found in [3]. Some temporal/time perspective approaches for information systems specifications and design are found in [34][28][7] and [32]. An analysis of three conceptual models with time perspective is found in [18]. Some temporal frameworks for database specification are presented in [8][14] and [33]. In [8] and [14], 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 [33], a set of static, dynamic and side-effect axioms are stated for maintaining the consistency of a database.

Our framework is similar to [8][14]. The difference is in the explicit specification of preconditions and the postconditions of operations. In this sense, our approach is in agreement with the opinion held by [2][6][35]. That is, a specification should specify the rules and assumptions explicitly and suppress exceptional details (when needed), in order to facilitate comprehension and change.

Formal methods for verifying information system/database specifications are found in [28][20][22][5]. The work in [5] 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. A method for verifying liveness of concurrent programs is found in [29] 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 [15] which contains a PROLOG implementation of some relevant parts.

## 3 DATABASE SPECIFICATION

### 3.1 Static Constraints

Static constraints of a database are specified as first order sentences. Examples of static constraints are:

$sc_1$: Every employee earns more than \$20,000.

$$(\forall x)(\forall y)(\ E(x,y) \longrightarrow y > 20{,}000\ )$$

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.

### 3.2 The Semantics

A database state is defined as a structure $S(U,I)$ of the first order language $L$, where $U$ is called the universe of the structure which is a non-empty set of individuals; $I$ is the interpretation of the structure which assigns an element of $U$ to each constant symbol, a mapping from $U^m$ 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 $S$. 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 model 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 $S_i \models SC$.

A result from model theory seems to be very useful for our purpose. It states that a set of wffs is consistent iff it has a model (the extended completeness theorem [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.

### 3.3 Temporal Constraints

Temporal constraints are defined in a temporal language. In the definition, 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 always← is read "always in the past excluding the present", and ever→' is read "ever in the future including the present." The other 6 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 §3.5.

```
<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←' | ever← | ever←'
<tqf> ::= always→ | always→' | ever→ | ever→'
<wff'> ::= wff in which free variables are
<parameter>s.
<parameter> ::= $\bar{x}$ | $\bar{y}$ | .. |
<function-symbol>(<parameter-list>)
<parameter-list> ::= <parameter> |
<parameter>,<parameter-list>
<function-symbol> ::= any function symbol of $L$.

The above definition is extended to include V and $\longrightarrow$ as usual.

In agreement with [14], we distinguish global and local quantifications. We use the concept of parameter. 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 a database state, 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 (E(x,y) is as defined in §3.1):

tc1: $E(\bar{x},\bar{y}) \longrightarrow$ always→$(\forall z)( E(\bar{x},z) \longrightarrow z \geq \bar{y} )$

The temporal constraint stating that "whoever has been an employee cannot be hired again" can be expressed as:

tc2: ever←' $(\exists y)E(\bar{x},y) \longrightarrow$ ~EXECUTABLE(hire($\bar{x}$))

where hire($\bar{x}$) is the operation of hiring $\bar{x}$, which is to be defined in the next subsection. 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.

## 3.4 The Semantics

Let $\sigma = \ldots S_{-1} S_0 S_1 S_2 \ldots$, denote a sequence of database states, where $S_0$ denotes the current state, $\ldots S_{-2}\ S_{-1}$ denotes the history, and $S_1 S_2 \ldots$ denotes the future of the database. Let $\sigma_j$ denote the sequence of states $\ldots\ S_{j-1} S_j$ or the sequence of states $S_j S_{j+1} \ldots$, depending on $j =< 0$ or $j >= 0$. Further, we use $\delta(<op>, S_i)$ to denote the state resulting from executing the operation <op> in state $S_i$. The semantics of the temporal assertions are as follows:

If w is not a temporal assertion, then $\sigma \models w$ iff $S_0 \models w$.
Now let w denote any temporal assertion, then
$\sigma \models$always←w iff $(\forall j<0)(\sigma_j \models w)$
$\sigma \models$always←'w iff $(\forall j=<0)(\sigma_j \models w)$
$\sigma \models$ever←w iff $(\exists j<0)(\sigma_j \models w)$
$\sigma \models$ever←'w iff $(\exists j=<0)(\sigma_j \models w)$
$\sigma \models$always→w iff $(\forall j>0)(\sigma_j \models w)$
$\sigma \models$always→'w iff $(\forall j>=0)(\sigma_j \models w)$
$\sigma \models$ever→w iff $(\exists j>0)(\sigma_j \models w)$
$\sigma \models$ever→'w iff $(\exists j>=0)(\sigma_j \models w)$
$\sigma \models \sim w$ iff not $\sigma \models w$
$\sigma \models$EXECUTABLE(<op>) iff the following conditions hold:
$\sigma \models$the temporal assertion of the operation <op>.
$S_0 \models$the precondition of the operation <op>.
$\delta(<op>, S_0) \models$the postcondition of <op>.

Let $w_1$ and $w_2$ be two temporal assertions: $\sigma \models (w_1 \& w_2)$ iff $\sigma \models w_1$ and $\sigma \models w_2$. The definition is similarly extended to include V and $\longrightarrow$.

We must define the semantics for $\sigma_j$, we have:

If w is not a temporal assertion, then $\sigma_j \models w$ iff $S_j \models w$.
If w is a temporal assertion, then
$\sigma_j \models$always←w iff j=<0 and $(\forall k<j)(\sigma_k \models w)$
$\sigma_j \models$always←'w iff j=<0 and $(\forall k=<j)(\sigma_k \models w)$
$\sigma_j \models$always→w iff j>=0 and $(\forall k>j)(\sigma_k \models w)$
$\sigma_j \models$always→'w iff j>=0 and $(\forall k>=j)(\sigma_k \models w)$
$\sigma_j \models$ever←w iff j=<0 and $(\exists k<j)(\sigma_k \models w)$
$\sigma_j \models$ever←'w iff j=<0 and $(\exists k=<j)(\sigma_k \models w)$
$\sigma_j \models$ever→w iff j>=0 and $(\exists k>j)(\sigma_k \models w)$
$\sigma_j \models$ever→'w iff j>=0 and $(\exists k>=j)(\sigma_k \models w)$
$\sigma_j \models \sim w$ iff not $\sigma_j \models w$
$\sigma_j \models$EXECUTABLE(<op>) iff
$\sigma_j \models$the temporal assertion of the operation <op>.
$S_j \models$the precondition of the operation <op>.
$\delta(<op>, S_j) \models$the postcondition of <op>.

If $w_1$ and $w_2$ are two temporal assertions: $\sigma_j \models (w_1 \& w_2)$ iff $\sigma_j \models w_1$ and $\sigma_j \models w_2$. The definition is similarly extended to include V and $\longrightarrow$.

## 3.5 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. Syntactically, an operation is defined (in BNF) 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'>, $S_i \models$<pre>
==> $\delta(<op>, S_i) \models$<post>
<tap'> ::= any | <tap>
<pre> ::= <wff'>
<post> ::= <wff'>

In the above definition, <parameter-list>, <tap>, and <wff'> are as in §3.3. <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. any is used as a dummy temporal assertion when there is no need to refer to the past.

The following examples serve to illustrate the specification of some operations. The meanings of the predicate symbols are as defined in §3.1.

$hire(\bar{x})$: always← $\tilde{}(\exists y)E(\bar{x},y)$, $S_i \models \tilde{}(\exists y)E(\bar{x},y)$
==> $\delta(hire(\bar{x}),S_i) \models (\exists y)(E(\bar{x},y)\&y>20000)$

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 $y>20000$." Note that the temporal assertion "always← $\tilde{}(\exists y)E(\bar{x},y)$" is required in this operation description because of tc2 in §3.3. We require that $y>20000$ because of sc1 in §3.1. Other operation descriptions are listed below:

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

$raise^*(\bar{x},10\%*\bar{y})$: any, $S_i \models E(\bar{x},\bar{y})$ ==>
$\delta(raise^*(\bar{x},10\%*\bar{y}),S_i) \models E(\bar{x},\bar{y}+10\%*\bar{y})\&\bar{y}+10\%*\bar{y}>20000$

$promote(\bar{x})$: any, $S_i \models (\exists y)E(\bar{x},y)$ & $\tilde{}M(\bar{x})$
==> $\delta(promote(\bar{x}),S_i) \models M(\bar{x})$

$demote(\bar{x})$: any, $S_i \models M(\bar{x})$
==> $\delta(demote(\bar{x}),S_i) \models \tilde{}M(\bar{x})$

$Engage(\bar{x})$: any, $S_i \models \tilde{}(\exists y)E(\bar{x},y)$ & $\tilde{}M(\bar{x})$
==> $\delta(engage(\bar{x}),S_i) \models M(\bar{x})$

Note that a postcondition in problem-solving of artificial intelligence may 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 [24][17]. 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 because a database specification is not meant to be an executable object. We will return to this point in §4.2.

## 4 VERIFICATION OF DATABASE SPECIFICATION

### 4.1 Verification of Static Constraints

We first briefly review the results from [21] which may be used to reduce the effort in using a theorem-prover, e.g., the resolution principle [31]. Let C be a set of clauses. The unifiability digraph $G(C)=<C,E>$ is a directed graph with C as its points and $<c_i,c_j> \in E$ if $c_i$ contains a positive literal which is unifiable with the negation of some negative literal in $c_j$ (for unification see [27][31]). If G(C) consists of several subdigraphs, $G(C_i)$, i = 1, 2, ..., then C is consistent iff each $C_i$ is consistent. If G(C) contains no point of indegree (outdegree) zero, then C is consistent. If G(C) has no semicycle and C contains no two clauses $c_1$, $c_2$ with some distinct literals $\alpha,\beta \in c_1$ and some distinct literals $\gamma,\eta \in c_2$, such that $\alpha$ is unifiable with $\tilde{}\gamma$ and $\beta$ is unifiable with $\tilde{}\eta$, then C is said to be compact. If C is compact, then the consistency of C is decidable, i.e., only finite many new clauses can be produced by unrestricted resolution [21].

We may now present the verification process:

1) Transform the set SC of static constraints into a set $C_{SC}$ of clauses. This can be easily done (see e.g., [27]).
2) Construct a unifiability digraph $G(C_{SC})$ for the set $C_{SC}$ of clauses obtained in 1). For each subgraph $G(C_i)$ of $G(C_{SC})$, where $C_i \subseteq C_{SC}$, perform the following steps. If each $C_i$ is consistent, so is $C_{SC}$.
3) According to Robinson's purity principle [31], we may remove any point of $G(C_i)$ which contains a literal not unifiable with the negation of any literal of $C_i$ without affecting the consistency of $C_i$. (This can be done more easily in a connection graph [16]). Repeatedly apply this principle until no clause can be removed. Denote the resulting digraph by $G(C_i')$.
4) From $G(C_i')$ determine if $C_i'$ is consistent.
5) If the consistency of $C_i'$ cannot be determined in 4), then use $G(C_i')$ to determine if $C_i'$ is a compact set of clauses. If so, then a bottom-up resolution process is stopped when the empty clause is generated (then $C_i'$ is inconsistent) or no new clauses can be generated (then $C_i'$ is consistent). Otherwise, certain time bound must be set depending on the theorem-prover used and the complexity of the input. When the bound is reached before any inconsistency is detected we take it for granted that $C_i'$ must be consistent. An alternative is to modify the static constraints so that the $C_i'$ resulting from 3) is either consistent or compact and hence its consistency can be formally checked.

Our example { sc1, sc2 } in §3.1 is consistent since both of them can be removed in step 3). Nontrivial examples can be found in [19].

### 4.2 Operation Analysis

#### 4.2.1 Informal Discussion of Basic Ideas

Suppose that we have a relational database with only one static constraint: sc1': $(\forall x)(\exists y)(\ M(x) \rightarrow E(x,y)\ \&\ y>20000)$ which says that every manager is an employee having some salary more than 20000. For simplicity, we assume that there are only one person identified by name n and two salary values $s<20000$ and $s'>20000$. It can be easily seen that S1, S2, S3 in Fig.1a are legal

database states. 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 | |

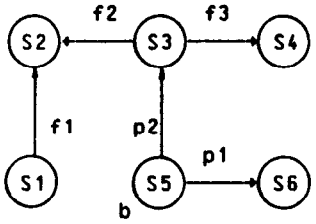



Fig. 1. a transition diagram for fire and promote

Transition f1 has the following properties: the operation is applicable in a legal database state; the application yields some legal database state; 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 <n,s'>∈E and <n>∉M. If transition f2 is wanted, then the operation description should contain in the "postcondition" that after executing the operation, <n,s'>∉E and <n>∉M.

Suppose that the relational database imposes one more static constraint sc2': (∀x)(∀y)( E(x,y) → y<20000 ) which says that every employee has salary less than 20000. 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 <n,s>∈E and <n>∉M then its application leads to the state in which <n>∈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, <n>∈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 the next section, we will present a method which will analyse each operation description to make sure that the operation is applicable, yielding legal states and the description is sufficient for characterizing the poststate.

### 4.2.2 Formal Analysis of Operation Descriptions

First notice that when we start to analyse the operations, we already have a consistent set SC of static constraints. Second, following Gallaire and Nicolas and Minker [13][26][25], we distinguish two kinds of static constraints: those that are used as integrity constraints (e.g., every manager is an employee) and those that are used as derivation rules (e.g., x>y & y>z → x>z).

In what follows, let P and Q denote the clause forms of the precondition and the postcondition of an operation <op> in question. Let SC and $SC_I \subseteq SC$ denote the set of static constraints and integrity constraints and let $C_{SC}$ and $C_I$ be the clause forms of SC and $SC_I$ respectively. We have the following

THEOREM 4.1. An operation is applicable in some legal database state iff P U $C_{SC}$ is consistent. (We put the proof in the appendix)

Let C be any set of clauses. Let $\Pi(C)$ denote the union of C with the set of all clauses that can be obtained by resolution between pairs of clauses in C. By $\Pi^2(C)$ we mean $\Pi(\Pi(C))$, and $\Pi^0(C) = C$. It can be proved that for all c for all k, $c \in \Pi^k(C)$ implies that c is a logical consequence of C. In particular, the set of literals logically following C is defined by $\psi(C)=\{\alpha : \alpha$ is a literal & $(\exists k)(\alpha \in \Pi^k(C))\}$. We have

THEOREM 4.2. Let $S_i$ be a legal database state and $\alpha$ any literal. If $\alpha \in \Pi^k(C_I \cup P)$ (resp. $\Pi^k(C_I \cup Q)$) for some k and $S_i \not\models P$ (resp., $S_i \not\models Q$), then $S_i \not\models \alpha$. (We put the proof in the appendix)

COROLLARY. If $C_I \cup P$ (resp. $C_I \cup Q$) is compact, then $\psi(C_I \cup P)$ (resp. $\psi(C_I \cup Q)$) is finite and hence $S_i$ is finite.

The operation analysis is performed in two main steps: I) The analysis of the insert and delete operations. II) The analysis of the update operations.

I) The analysis of the insert and delete operations: If <op> denotes an insert or delete operation, then

i) <op> is applicable in some legal database state iff P U $C_{SC}$ is consistent (Theorem 4.1). If so, the prestate of <op> is characterized by the set (Theorem 4.2)

prestate(<op>) = $\psi(C_I \cup P)$

This set is not a complete representation of the prestate but only a characterization of a relevant part of a legal database state such that the integrity constraints are true

in it. The characterization of the prestate is similar to the use of the integrity constraints as generation rules under a closed world assumption (conf. [26]). Furthermore, if $C_I UP$ is compact, then $\psi(C_I UP)$ is finite. Otherwise, we have to use some of the $SC_I$ as derivation rules in order to gaurantee that $S_i$ is finite (Theorem 4.2).

ii) The application of <op> yields some legal database state iff poststate(<op>) U $C_{SC}$ is consistent, where

$$\text{poststate}(<op>) = \{ \alpha : \alpha \in \psi(QU\Phi(<op>)) \} \cup \Phi(<op>) \text{ and}$$

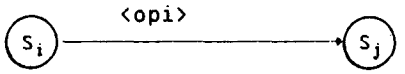
$$\Phi(<op>) = \{\alpha: \alpha \in \psi(PUC_I) \& \{\alpha\}UQ \text{ is consistent}\}$$

Intuitively, $\Phi$(<op>) denotes those literals that are true in the prestate and are not falsified by (the postcondition of) <op>. Note that the frame problem is implicitly treated in this step.

iii) The operation description is sufficient for characterizing the poststate if $\psi(C_I UQ) \subseteq$ poststate(<op>). (Theorem 4.2)

iv) If one of the above test fails, we must modify the static constraints and/or the operation description and repeat the whole process until each operation description passes the above tests.

v) At this stage, we should have a list of state transitions each of which can be depicted as




prestate(<opi>) poststate(<opi>)

where $S_i$, $S_j$ are the (unique) names given to the prestate and the poststate of the operation <opi>.

vi) We now merge some of the prestates and/or poststates as follows†:

a) Let $S_k$ be a presetate and $S_l$ be a prestate or a poststate. If $S_k \subseteq S_l$ (improper set inclusion), merge $S_k$ into $S_l$ (i.e., draw an arc from $S_l$ to $S_m$ for each arc from $S_k$ to $S_m$ for all m and label the arc accordingly; remove $S_k$ along with its arcs). This step is repeated until no merge is possible.

b) Let $S_k$, $S_l$ be two poststates. If $S_k = S_l$ merge $S_k$ into $S_l$. Repeat this step until no merge is possible.

† Sometimes we have to rename the parameters and/or the variables, however we cannot discuss this here due to space limit.

vii) At this stage, we should have a transition diagram for the insert and delete operations.

Fig.2 shows the result of analysing the operation descriptions defined in §3.5.

| | prestate | poststate |
|---|---|---|
| hire | $S_1 = \{\tilde{}E(\bar{x},y), \tilde{}M(\bar{x})\}$ | $S_2 = \{E(\bar{x},f(\bar{x})), f(\bar{x})>20000, \tilde{}M(\bar{x})\}$ |
| fire | $S_3 = \{E(\bar{x},f(\bar{x})), f(\bar{x})>20000, \tilde{}M(\bar{x})\}$ | $S_4 = \{\tilde{}E(\bar{x},y), f(\bar{x})>20000, \tilde{}M(\bar{x})\}$ |
| pro-mote | $S_5 = \{E(\bar{x},f(\bar{x})), f(\bar{x})>20000, \tilde{}M(\bar{x})\}$ | $S_6 = \{E(\bar{x},f(\bar{x})), f(\bar{x})>20000, M(\bar{x})\}$ |
| de-mote | $S_7 = \{E(\bar{x},f(\bar{x})), f(\bar{x})>20000, M(\bar{x})\}$ | $S_8 = \{E(\bar{x},f(\bar{x})), f(\bar{x})>20000, \tilde{}M(\bar{x})\}$ |

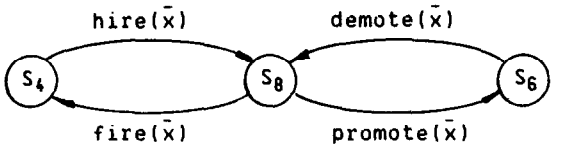



Fig. 2. The transition diagram for the insert and delete operations

Note that the engage operation cannot pass substep ii) and we have chosen to remove the engage operation description from our study.

II) In a similar way, we may analyse the update operations. The result is shown in Fig.3. The detailed steps and the interpretation of Fig.3 can be found in [19].

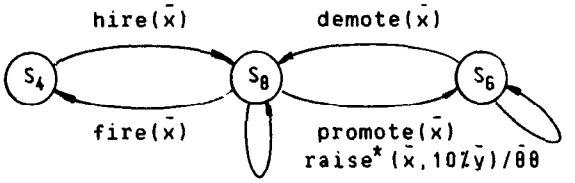



Fig. 3. The transition diagram of the example

## 4.3 Verification of Temporal Constraints

The basic ideas and steps for verifying the temporal constraints can be informally stated as follows ([19] gives the details):

1) Transform the transition diagram of the last section into a family of finite automata $fa_i$. This is done by taking in turn each state of the transition diagram as the initial state and let every state be a final state. $S_i$ is interpreted as the current database state. The $fa_i$'s are used to generate test sequences and construct the pushdown automata $pa_i$'s.

2) Generate test sequences: There is a method for generating a set of test sequences for any given finite automaton with initial state

$S_i$ [10]. Let $\tau_i$ denote the set of test sequences that is generated for $fa_i$. Then for every transition from $S_{j1}$ to $S_{j2}$ on $\langle op_k \rangle$, there are sequences $w$ and $w\langle op_k \rangle$ in $\tau_i$ such that $w$ forces $fa_i$ into $S_{j1}$ from its initial state $S_i$. Further, every $w \in \tau_i$ is of length less than or equals to n, where n is the number of states of $fa_i$.

3) Construct $pa_i$'s: By definition, a temporal constraint specifies some condition which must be satisfied by every permissible sequence of operations. A sequence of operations is a permissible sequence if it is permitted, by the operation descriptions, to be executed on the database, which is in some current state $S_i$. To analyse whether a test sequence is a permissible sequence, we transform the $fa_i$ into a pushdown automaton $pa_i$ where the pushdown "symbols" of $pa_i$ are temporal assertions. The top element of the pushdown store is intended to represent the database history before and excluding the current state at all time. At each step, the $pa_i$ examines if the stack top implies the temporal assertion of the operation in question. If so, it enters a new state and tries to remember that something has happened by pushing a temporal assertion onto the stack. This temporal assertion thus represents the new database history and it is obtained by considering the preceding database history and the precondition of the operation in question. A test sequence is permitted to be executed in the database state $S_i$ iff it is accepted by $pa_i$.

4) Verify temporal constraints: For each test sequence $\langle op_1 \rangle \ldots \langle op_m \rangle \in \tau_i$ that is accepted by $pa_i$, there is a sequence of states $S_i S_{i+1} \ldots S_{i+m}$ such that $S_{i+j}$ is entered from $S_{i+j-1}$ on $\langle op_j \rangle$ in $fa_i$, $j=1,\ldots,m$. We call such a sequence of states a partial execution sequence since it represents only the future seen from $S_i$ or only the past seen from $S_{i+m}$. The set of all partial execution sequences over all i is denoted by $\Sigma'$. It is an easy task of forming the set $\Sigma$ of total execution sequences. If $wS_j$, $S_j\eta$ are in $\Sigma'$, then $wS_j\eta$ is in $\Sigma$: $S_j$ is interpreted as the current state, $w$ the history, and $\eta$ the future of the database. We see that each element of $\Sigma$ is of the form $\ldots S_{-1} S_0 S_1 \ldots$ which we have used to define the semantics of the temporal language in §3.4. Thus, the verification of the temporal constraints is to verify that each element of $\Sigma$ satisfies each of the temporal constraints according to the semantics defined in §3.4.

## 5 CONCLUDING REMARKS

In this paper, we have presented a temporal framework for database specification and verification. The specification of databases is declarative and application oriented. For simplicity, we have chosen in this paper to specify the static constraints in the first order logic. In fact, many-sorted logic could have been used instead. In this case, the unification algorithm will be slightly different, see e.g. [30]. Further extention of the temporal language that is used to specify the temporal constraints and the temporal assertions of the operations is possible. For example, we may include tense logic operators such as "before", "after", "at" etc. [34]. We want to stress that queries to the database can be expressed in the framework by using set notions and set operations, although we did not discuss this aspect in the paper.

The verification of a database specification can be supported by existing theorem-proving mechanisms, e.g., the resolution principle. In this paper, we have chosen to present the verification framework in a straight way without considering efficient strategies for carrying out the tasks. In practice, heuristic information and more intelligent strategies should be used [27][17].

## 6 ACKNOWLEDGMENT

The author wants to thank Prof.Solvberg for his encouragement and constructive discussions. Thanks to Tore Amble for informal discussions and to Even Johansen for making the macros for the mathematical symbols.

REFERENCES

[1] Anderson, T.L., The Database Semantics of Time, Doctoral Thesis, Univ. of Washington, Jan., 1981.

[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] Bolour, A. and L.J. Dekeyser, Abstractions in temporal information, in Inform.Syst., vol.8, no.1, 1983. pp.41--49.

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

[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] Connor, M.F., Structured analysis and design technique, in Systems Analysis and Design, A Foundation for the 1980's, Edited by Cotterman, W.W. et al, N.H. Publ. Comp., 1981. pp.213--234.

[12] Furtado, A.L., Dynamic modelling of a simple existence constraint, in Information Systems, Vol.6, 1981. pp.73--80.

[13] Gallaire, H. and J.Minker, Logic and Data Bases, Plenum Press, N.Y., 1978.

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

[15] Hove, J.O., Kungs Metode for Konsistensbevis og Modellkonstruksjon Anvendt pa Kommunikasjons Protokaller, Master Thesis, Dept. of Computer Science, The Norwegian Inst. of Tech., Trondheim, NORWAY, 1984.

[16] Kowalski, R., A proof procedure using connection graphs, JACM, Vol.22, 1975. pp.572--595.

[17] Kowalski, R., Logic for Problem Solving, Elseview, N.H., Inc., 1979.

[18] Kung, C.H, An analysis of three conceptual models with time perspective, in Information Systems Design Methodologies: A Feature Analysis, Olle et al (ed.s), N.H. Publ. Comp., 1983. pp.141--168.

[19] Kung, C.H, A Temporal Framework for Information Systems Specification and Verification, Doctoral Thesis, Dept. of Computer Science, The Norwegian Inst. of Tech., Trondheim, NORWAY, 1984.

[20] Leveson, N.G., A.I. Wasserman, and D.M. Berry, BASIS: A behavioral approach to the specification of information systems, in Information Systems, Vol.8, No.1, 1983. pp.15--23.

[21] Lewis, H.R., Cycles of Unifiability and Decidability by Resolution, Aiken Computation Laboratory, Harvard Univ., Tech. Rept., 1975.

[22] Lundberg, B., IMT - An information modelling tool, Proc. IFIP WG8.1 WC on Automated Tools for IS Design and Development, New Orleans, USA, 1982.

[23] Lundberg, B., An axiomatization of events, BIT 22, 1982.

[24] Mcarthy, J. and P. Hayes, Some philosophical problems from the stanpoint of artificial intelligence, in Machine Intelligence no.4, B.Meltzer and D.Michie (ed.s), Edinburgh Univ. Press, Edinburgh, 1969. pp.463--502.

[25] Minker, J. and J.M. Nicolas, On recursive axioms in deductive databases, Inform.Syst., vol.8, no.1, 1983. pp.1--13.

[26] Nicolas, J.M. and K. Yazdaman, Integrity checking in deductive databases, in Logic and Databases, Gallaire and Minker editors, Plenum Press, N.Y. 1978. pp.325--346.

[27] Nilsson, NILS J., Problem-Solving Methods in Artificial Intelligence, McGraw-Hill Book Comp., 1971.

[28] Olive, A., Information derivability analysis in logical information systems, CACM Vol.26, No.11, Nov., 1983. pp.933--938.

[29] Owicki, S. and L. Lamport, Proving liveness properties of concurrent programs, ACM Trans. on Prog. Lang. and Syst., vol.4, no.3, July 1982. pp.455--495.

[30] Reiter, R., An Approach to Deductive Question-Answering, BBN Report No.3649, Cambridge, Mass., Sept. 1977.

[31] Robinson, J.A., A machine-oriented logic based on the resolution principle, J.ACM, vol.12, no.1, Jan. 1965. pp.23--41.

[32] Rolland, C., S. Leifert and C. Richard, Tools for information system dynamics management, Proc. 5th Intl' Conf. on VLDB, Oct. 1979. pp.251--261.

[33] Schiel, U., An abstract introduction to the temporal-hierarchic data model (THM), Proc. 9th Intl' Conf. on VLDB, Florence, Italy, Oct.31--Nov.2, 1983. pp.322--330.

[34] Sernadas, A., Temporal aspects of logical procedure definition, Info.Syst. Vol.5, No.3, 1980. pp.167--187.

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

APPENDIX:

PROOF (OF THEOREM 4.1). Suppose that P is true in a legal database state $S_i$, i.e., $S_i \models P$. Since $S_i$ is legal, i.e., $S_i \models C_{SC}$. Therefore, $S_i \models (PUC_{SC})$ and hence $PUC_{SC}$ must be consistent. Now suppose that $PUC_{SC}$ is consistent and let $S_j$ be a model of $PUC_{SC}$. We have $S_j \models C_{SC}$ and hence $S_j$ is legal. However, since $S_j \not\models P$ and hence the operation is applicable in some legal database state. E.Q.

PROOF (OF THEOREM 4.2). We prove the case for the precondition. Suppose that $S_i$ is a legal database state and $S_i \not\models P$. Since $S_i$ is legal, i.e., $S_i \models C_{SC}$ which implies that $S_i \models C_I$. Thus, $S_i \models (PUC_I)$. Since $\alpha \in \Pi^k(C_I \cup P)$ implies that $\alpha$ is a logical consequence of $C_I \cup P$. This implies that every model of $C_I \cup P$ is a model of $\alpha$ and hence $S_i \models \alpha$. E.Q.

# Session C2B

## Panel — Statistical Databases

Chairpersons

Z.M. Ozsoyoglu
U.S.A.

G. Ozsoyoglu
U.S.A.

**Proceedings of the Tenth International Conference on Very Large Data Bases**

**Singapore, August, 1984**