Handling Database Updates in Two-dimensional Temporal Logic

Marcelo Finger*†

Department of Computing, Imperial College
180 Queen's Gate
London SW7 2BZ, UK

E-mail:*mf3@doc.ic.ac.uk*

February 17, 1993
(Draft version 0.2)

Abstract

This paper deals with the description of the evolution of the understanding of the history of a particular world. We have particular interest in describing certain problems that occur in database systems due to updates. For this purpose, we introduce a two-dimensional temporal logic as a formalism which enables the description of both the history and the evolution of the beliefs about the history. The historical dimension describes the history of the world according to a certain belief. The belief dimension describes the evolution of those beliefs.

The historical dimension is then associated with an historical database, and transactions in the database are associated with changes in belief. Besides describing the history of the world, the database system can also execute temporal specifications in the form of temporal actions. The two-dimensional formalism is used to describe the effects of updates in the execution of temporal actions. Those descriptions are then taken as specifications for a database system and their implementation is discussed.

Keywords: Historical Databases, Temporal Logic, Temporal Specifications.

*I am very grateful to Prof. D. M. Gabbay for his helpful comments.

†Esprit BRA project 3096 (SPEC).

1 Introduction

This paper deals with the problem of describing the evolution of the understanding of the whole history of the world. The world is any particular object in a certain environment that we want to describe. Normally, the elements of the world are evolving systems, i.e. they cannot be isolated from the flow of time, so that the real object of our description becomes the history of the world. The particular evolving system that we are interested in describing is an historical database.

Conceptually, we have to bear in mind two distinct entities that we want to treat differently:

a) The world being described and its history.

b) The reference according to which the description is made. The reference is an observer looking at the history of the world. The reference is not part of the world.

The exclusion of the reference from the world is to avoid the possibility of having an observer observing himself, which may be a very interesting problem from the philosophical point of view, but it is not very useful in the database case where, in fact, we want to ensure the separation between the agent and the object.

A change in the world is seen as part of its history. A change in the reference is seen as a change in the observer's belief about the history of the world. If we further assume that the observer can act upon the world according to what he believes about the history of the world, we have these two entities constantly interacting with each other. The problem is describing those two constantly evolving entities and their constant interaction.

Instants of time also have a distinct treatment whether they are associated with the history or with a belief. An instant in time associated with the historical description of the world is called an evaluation point. An instant in time associated with the reference of a belief is called a reference point or a reference time.

If we apply these concepts to the description of databases, the history of the world is modelled by an historical database [Snodgrass & Ahn 85], wherein every piece of information is associated to an interval of time, called its validity interval. The actions developed upon the world are caused by temporal specifications of the form

$$condition \rightarrow action$$

where *condition* is verified against the history of the world (database), and *action* is to be executed upon the world in the present or future time. The idea of executing a temporal specification reflects an imperative view of time and is discussed in [Gabbay 89].

In this context, the reference can be seen as an observer that has a current belief about the history of the world according to which he checks the *condition* part of a temporal specification that, when verified, triggers the execution of *action* that will

influence the present and the future of the world. The observer is embedded in the database environment as the executing mechanism of temporal specifications.

The database user is able to add, delete and modify information in the database, changing the current view about the history of the world. The user can be then seen as some kind of "consciousness" operating behind the observer's "mind", causing the observer to change his beliefs about the history of the world. The observer's "mind" is changed whenever the user updates the database.

In order to deal with both an evolving history and evolving beliefs about it, we introduce a two-dimensional temporal logic.

Two-dimensional temporal systems have been used in the literature in order to capture some temporal constructions of natural language, such as the past perfect and the present perfect in [Gabbay 90]. An attempt to fomalise the meaning of verbs like "believe" and "say" in the temporal logic GR [Gabbay & Rohrer 78] is perhaps closer to our goals here. In such a logic, phrases like

Peter says it is raining.
Peter believes it is raining.

are captured in formulae of the form

$$E(x, q)$$
$$G(x, q)$$

where E and G are intended to capture the meaning of the verbs "say" and "believe", respectively, x is a variable that is associated with the person that said or believed (in this case: Peter), and q is the proposition representing what was said or believed (in this case: it is raining).

It is remarkable that, regarding the semantics of the temporal logic GR, the truth value of formulae like $E(x, q)$ and $G(x, q)$ is completely independent from the truth value of q. In other words, what Peter says or believes is absolutely independent, at least in principle, of what is happening in the "real world".

In the database case, there is no consideration about what the "real world" is meant to be. The presence of some data in the database is the condition for the observer to believe in it. If any information is deleted from the database, it becomes part of the observer's past beliefs. The fact that the database is supposed to represent with some accuracy a "real world" is irrelevant for its description, although it is not irrelevant for its proper use.

In order to illustrate the kind of problems we have in mind, let's consider the following example:

Example 1.1 *Retroactive payment*

> Suppose a database is used to record the information concerning the employees of a company, including their salaries. Suppose this information is used for the automatic printing of payment cheques on the eve of payment days. This activity can be modelled by the following temporal specification, written in a Prolog-like notation:

$$T\,payment\_day \wedge$$
$$employee(Person) \wedge$$
$$salary(Person, Amount)$$
$$\rightarrow print\_cheque(Person, Amount)$$

$T\,payment\_day$ means that tomorrow is the payment day. Time is discretely counted in days and the rule above is checked against the database every day. When it is the eve of the payment day (i.e. when the expression $T\,payment\_day$ evaluates to $true$), then every employee of the company has his salary searched in the database, and a command is issued to the printer to print a cheque with the name of the employee and his or her respective salary. This is what is meant by executing the temporal specification above.

But it may happen that a particular employee, say Mr. Jones, receives a retroactive increase of his salary after the printing of his payment cheque. At the payment day's eve, say 26/10/90, the piece of information present at the database was:

salary(Jones, \$1000) validity: 01/89 – now

which caused a cheque to be printed to Mr. Jones with the amount of \$1000 on it. But the retroactive increase of Mr. Jones's salary at 31/10/90 left the database with the following information:

salary(Jones, \$1000) validity: 01/89 – 09/90
salary(Jones, \$1200) validity: 10/89 – now

If this information was present in the database at the moment Mr. Jones's \$1000 payment cheque was printed, a \$1200 cheque would have been printed instead. This means that, after the change, the database views the printing of the \$1000 payment cheque as based on false information. We say that, in this situation, the executed action has become non-supported, as defined in [Finger 90]. At this point, it would be necessary to take some kind of corrective action in order to deal with the effects of the presence of the non-supported action in the system. We would like the database system to detect the presence of such non-supported action, and we may leave the responsibility of taking corrective actions either with the database user or we may wish to specify corrective actions that should be executed when a non-supported action is detected.

We are going now to present the two-dimensional temporal logic, built to deal with these problems.

2 Propositional Two Dimensional Temporal Logic

We present here a two-dimensional system that enables us to reason about the history of the world as well as to reason about the evolution of beliefs about the

world. Let us first consider informally the temporal operators U and S — until and since — introduced by Kemp and discussed in [Gabbay 81], as well as the less expressive operators that can be derived from U and S, namely P (past), F (future), H (always in the past), and G (always in the future). These operators allow us to reason about the history of the world. The operators U and S are two-place; P, F, H and G are one-place operators; so that, if A and B are correct formulae of our language (yet to be defined) describing the world, we can build, by using the operators, other formulae like FA and $S(A, B)$ with the following intended meaning:

- FA means that there is a time point in the future where A holds in it.
- $S(A, B)$ means that there is a time point in the past where A holds in it and B holds in all time points since then.

Recall that we have already established a distinction between evaluation time points and reference time points. The operators U, S, P, F, H and G always refer to the evaluation time points because they are associated to the historical description of the world.

The next step is to provide a set of operators to describe beliefs about the world in an independent way. We do this by duplicating the set of temporal operators so that we have a new family of operators specialized in expressing beliefs about the history. This family is called the $\mathcal{T}$-operators (or belief operators) and is composed by $\mathcal{T}_U$, $\mathcal{T}_S$, $\mathcal{T}_P$, $\mathcal{T}_F$, $\mathcal{T}_H$ and $\mathcal{T}_G$.

If A and B are formulae of our language, the intended reading of the $\mathcal{T}$-operators is the following:

- $\mathcal{T}_P A$ means that there was a reference point in the past where A was believed to hold in it;
- $\mathcal{T}_F A$ means that there will be a reference point in the future where A is believed to hold in it;
- $\mathcal{T}_H A$ means that A has always been believed in the past instants;
- $\mathcal{T}_G A$ means that A will always be believed in the future instants;
- $\mathcal{T}_S(A, B)$ means that B has been believe since A was believed
- $\mathcal{T}_U(A, B)$ means that B will be believed until A is believed.

We are aware that the intended meaning presented above is certainly not free from ambiguities, since the natural language statement "is believed" does not clearly establish, in formulae like $\mathcal{T}_P A$, in which point of time A is going to be evaluated. This problem can only be solved by giving a formal semantic definition, which will be done later in this section.

Note that, in principle, we are dealing with two different flows of time, that can even have different natures. For instance, we may want to describe the history of the world in a dense flow of time that allows us always to refine a certain description

of the history of the world in terms of temporal details. On the other hand, we may want to talk about a succession of beliefs about this history, where a change in beliefs is triggered by some event, e.g., the learning of some new information. In this case, the flow of belief time is clearly discrete, therefore we can have a dense flow of time in the historical dimension and a discrete flow of time in the belief dimension. This means that we can have two completely independent flows of time being dealt with by the same logical system. In this presentation we are going to consider two independent linear and discrete flows of time, so that we will be able to define the operator T (for tomorrow), Y (for yesterday), $\mathcal{T}_T$ (for the next belief) and $\mathcal{T}_Y$ (for the previous belief).

While defining the language on which the two-dimensional logic is going to be based, attention must be paid in order to maintain the separation between the history, in one dimension, and the beliefs about it, in the other dimension; furthermore, as stated before, the observer must not be considered as part of the world. We start by considering only a subset of all the possible two-dimensional formulae that can be generated, by the operators we have introduced, namely the belief formulae, and we present a semantics to those formulas. The semantics will shed light to our understanding of the operators, so that we will be able to extend the interpretation given to the $\mathcal{T}$-operators, and hence extend the two-dimensional language to include many other formulae that were not meaningful under the original interpretation.

2.1 Historical and belief formulae

Having in mind the intended meaning for the $\mathcal{T}$-operators as presented above, consider A as a formula describing the world. Then we have as perfectly meaningful formulae: PA, $\mathcal{T}_P A$ and $\mathcal{T}_P PA$. Nevertheless, the formula $P\mathcal{T}_P A$ seems meaningless, i.e. it cannot be read in a meaningful way according to the given interpretation to the operators; furthermore it appears to show a reference being described as part of the world, since a belief operator is inside the scope of an historical operator. In order to rule out this possible kind of formula, the formulae of our language are presented in two steps, namely historical formulae and belief formulae. Let us then formalise the language and its semantics.

Definition 2.1 *The alphabet*

The two-dimensional propositional alphabet is composed by:

- a countable set of propositional letters $\mathcal{L} = \{p_1, p_2, \ldots\}$;
- the boolean connectives $\neg$, $\vee$, $\wedge$ and $\rightarrow$;
- the two-place operators S, U, $\mathcal{T}_S$ and $\mathcal{T}_U$;
- punctuation signs (,) and ,. □

Note that the one-place operators P, F, H and G are not included in the alphabet since they can be syntactically defined in terms of U and S. The same consideration is valid for $\mathcal{T}_P$, $\mathcal{T}_F$, $\mathcal{T}_H$ and $\mathcal{T}_G$ with respect to $\mathcal{T}_S$ and $\mathcal{T}_U$.

Definition 2.2 *Historical formulae*

The set $\mathcal{H}_{\mathcal{F}}$ of historical formulae (or h-formulae) is the smallest set such that:

- every propositional letter belongs to it, and it is called an atomic formula;
- if $A \in \mathcal{H}_{\mathcal{F}}$ and $B \in \mathcal{H}_{\mathcal{F}}$ then $\neg A \in \mathcal{H}_{\mathcal{F}}$, $(A \wedge B) \in \mathcal{H}_{\mathcal{F}}$, $(A \vee B) \in \mathcal{H}_{\mathcal{F}}$ and $(A \rightarrow B) \in \mathcal{H}_{\mathcal{F}}$;
- $A \in \mathcal{H}_{\mathcal{F}}$ and $B \in \mathcal{H}_{\mathcal{F}}$ then $S(A, B) \in \mathcal{H}_{\mathcal{F}}$ and $U(A, B) \in \mathcal{H}_{\mathcal{F}}$. □

What we call here h-formulae is simply the language of the (one-dimensional) temporal logic of since and until. Those formulae are intended to describe only the history of the world. We do not write sometimes the outermost parentheses of a formula when no ambiguity is caused.

Definition 2.3 *Formulae containing an operator*

Let X be a two-place operator and let A and B be formulae. The set $\mathcal{F}_{\mathcal{X}}$ of formulae containing the operator X is the smallest set such that:

- $X(A, B) \in \mathcal{F}_{\mathcal{X}}$
- if $A \in \mathcal{F}_{\mathcal{X}}$ then $\neg A \in \mathcal{F}_{\mathcal{X}}$
- if at least one of A or B belong to $\mathcal{F}_{\mathcal{X}}$ then so do $A \wedge B$, $A \vee B$ and $A \rightarrow B$.

A formula A is said to contain the operator X iff $A \in \mathcal{F}_{\mathcal{X}}$. □

We are now in a position to define the set of belief formulae.

Definition 2.4 *Belief formulae*

The set $\mathcal{B}_{\mathcal{F}}$ of belief formulae (or b-formulae) is the smallest set such that:

- every h-formula belongs to it, and is called a current belief formula;
- if $A \in \mathcal{B}_{\mathcal{F}}$ and $B \in \mathcal{B}_{\mathcal{F}}$ then $\mathcal{T}_S(A, B) \in \mathcal{B}_{\mathcal{F}}$ and $\mathcal{T}_U(A, B) \in \mathcal{B}_{\mathcal{F}}$;
- if $A \in \mathcal{B}_{\mathcal{F}}$ so that A contains $\mathcal{T}_S$ or $\mathcal{T}_U$, then $\neg A \in \mathcal{B}_{\mathcal{F}}$;
- if $A \in \mathcal{B}_{\mathcal{F}}$ and $B \in \mathcal{B}_{\mathcal{F}}$ so that at least one of them contains $\mathcal{T}_S$ or $\mathcal{T}_U$, then $(A \wedge B) \in \mathcal{B}_{\mathcal{F}}$, $(A \vee B) \in \mathcal{B}_{\mathcal{F}}$ and $(A \rightarrow B) \in \mathcal{B}_{\mathcal{F}}$. □

The additional condition in the last two items of the definition of b-formulae are so as to provide a unique parsing to a formula; for a detailed discussion of this topic refer to [Finger 90].

We are now in a position to define the semantics for the propositional two-dimensional temporal language we have just presented.

Definition 2.5 *Two-dimensional valuations and models*

Consider a discrete flow of time composed by the pair $(\mathcal{T}, <)$, where $\mathcal{T}$ is a denumerable set of time points and $<$ is a transitive linear and irreflexive relation over $\mathcal{T}$. We will distinguish between the historical flow of time $(\mathcal{T}_h, <_h)$ and the reference (or belief) flow of time $(\mathcal{T}_b, <_b)$.

Let g be a function that for every reference time point $o \in \mathcal{T}_b$ and for every evaluation time point $t \in \mathcal{T}_h$ associates a set of propositional letters, i.e. $g(o,t) \subseteq \mathcal{L}$. The function g is called a two-dimensional valuation.

A two-dimensional model is a 7-tuple $\mathcal{M} = (\mathcal{T}_h, <_h, \mathcal{T}_b, <_b, o, t, g)$, where $(\mathcal{T}_h, <_h)$ is an historical flow of time, $(\mathcal{T}_b, <_b)$ is a reference flow of time, $o \in \mathcal{T}_b$ is the current belief time, $t \in \mathcal{T}_h$ is the current evaluation time and g is a two-dimensional valuation. □

Definition 2.6 *Semantics of b-formulae*

Let {0,1} be the set of truth values. Let $\mathcal{M} = (\mathcal{T}_h, <_h, \mathcal{T}_b, <_b, o', t', g)$ be a two-dimensional model. The notation $\| A \|^{\mathcal{M}}(o,t)$ reads the truth value of A at the evaluation point t with respect to the reference point o, in the model $\mathcal{M}$. We omit $\mathcal{M}$ when no confusion is caused, writing only $\| A \|(o,t)$.

Let A and B be b-formulae. We can then define $\| A \|(o,t)$ inductively:

1. $\| p_i \|(o,t) = 1$ iff $p_i \in g(o,t)$, p_i is an atomic formula;
2. $\| \neg A \|(o,t) = 1$ iff $\| A \|(o,t) = 0$;
3. $\| A \wedge B \|(o,t) = 1$ iff $\| A \|(o,t) = 1$ and $\| B \|(o,t) = 1$;
4. $\| A \vee B \|(o,t) = 1$ iff $\| A \|(o,t) = 1$ or $\| B \|(o,t) = 1$;
5. $\| A \rightarrow B \|(o,t) = 1$ iff $\| A \|(o,t) = 0$ or $\| B \|(o,t) = 1$;
6. $\| S(A,B) \|(o,t) = 1$ iff $\exists s(s <_h t \wedge \| A \|(o,s) \wedge \forall u(s <_h u <_h t \rightarrow \| B \|(o,u) = 1))$;
7. $\| U(A,B) \|(o,t) = 1$ iff $\exists s(t <_h s \wedge \| A \|(o,s) \wedge \forall u(t <_h u <_h s \rightarrow \| B \|(o,u) = 1))$;
8. $\| \mathcal{T}_S(A,B) \|(o,t) = 1$ iff $\exists x(x <_b o \wedge \| A \|(x,t) \wedge \forall y(x <_b y <_b o \rightarrow \| B \|(y,t) = 1))$;
9. $\| \mathcal{T}_U(A,B) \|(o,t) = 1$ iff $\exists x(o <_b x \wedge \| A \|(x,t) \wedge \forall y(o <_b y <_b x \rightarrow \| B \|(y,t) = 1))$; □

Note that, since h-formulae constitute a subset of b-formulae, items 1–7 of the definition above also give the semantics of h-formulae.

Once we have defined the syntax and semantics for the present two-dimensional language, we can extend it to by defining one-place operators in terms of the two-place operators.

Definition 2.7 *One-place temporal operators*

Consider *true* to represent $p \rightarrow p$, i.e. a formula that always evaluates to 1; and consider $false$ to represent $p \wedge \neg p$, i.e. a formula that always evaluates to 0. Let A be ah h-formula and B be a b-formula. We can then define:

1. $FA =_{def} U(A, true)$;
2. $PA =_{def} S(A, true)$;
3. $GA =_{def} \neg F \neg A$;
4. $HA =_{def} \neg P \neg A$;
5. $TA =_{def} U(A, false)$;
6. $YA =_{def} S(A, false)$;
7. $\tau_F B =_{def} \tau_U(B, true)$;
8. $\tau_P B =_{def} \tau_S(B, true)$;
9. $\tau_G B =_{def} \neg \tau_F \neg B$;
10. $\tau_H B =_{def} \neg \tau_P \neg B$;
11. $\tau_T B =_{def} \tau_U(B, false)$;
12. $\tau_Y B =_{def} \tau_S(B, false)$. □

We can then derive the semantic definition of $\tau_F A$, for instance, which has the intended meaning that, in some future belief time, A will be believed to hold:

$$\begin{aligned} \| \tau_F A \|(o,t) = 1 \text{ iff } & \tau_U(A, true) \\ \text{iff } & \exists x(x <_b o \wedge \| A \|(x,t) = 1 \wedge \\ & \quad \forall y(x <_b y <_b o \rightarrow \| true \|(y,t) = 1)) \\ \text{iff } & \exists x(x <_b o \wedge \| A \|(x,t) = 1) \end{aligned}$$

Note that the evaluation point t is not affected by the evaluation of $\tau_F A$. In fact, the evaluation point t is not affected by the semantics of either τ_U or τ_S or any of their derived operators. As a consequence, it is irrelevant for their semantic definition the relative position between the reference and the evaluation points. But this is not always the case for all two-dimensional logics (see, for instance, the F^*P^* two dimensional system defined in [Gabbay 90]). The temporal operators can be seen as performing "moves" or "jumps" in time. The application of a temporal operator moves either the evaluation point or the reference point or even both. In this sense, consider the situation ilustrated in Figure 1.

Figure 1a illustrates the kind of movement caused by any of the τ-operators, carrying the reference time but leaving the evaluation time in the same position. On the other hand, the historical operators, can be seen as moving only the evaluation point while leaving the reference point unchanged, as illustrated in Figure 1b.

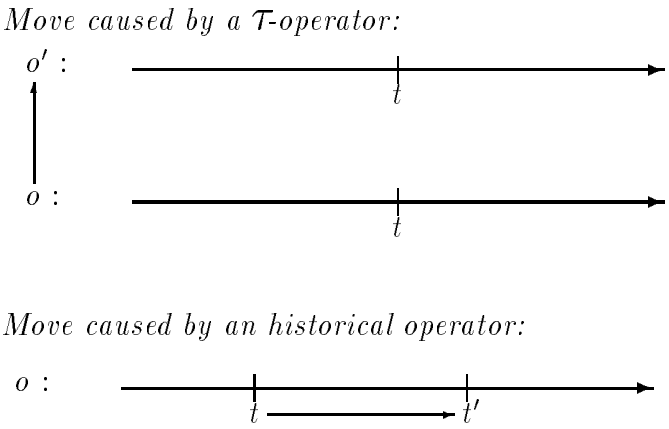


Figure 1: Time moves caused by temporal operators

2.2 Two-dimensional formulae

In the semantics definition of b-formulae we see that the historical and belief formulae are completely orthogonal, in a sense that an historical operator provokes no change in the belief dimensions and that a belief operator provokes no change in the historical dimension. This suggests a reinterpretation for these operators as mere displacement operators. An operator like P should be read as "a displacement towards the past in the historical dimension" a formula like $\mathcal{T}_S(A, B)$ should be read as "in the present evaluation time, A was a past belief and B has been believed since". Due to the orthogonality between historical and belief operators, there is no way of having a reference as part of the world nor having a belief as part of the history.

Under this new interpretation, a formula like $P\mathcal{T}_P A$, which is not a b-formula, is acceptable and reads "going to the past in history then going to a past belief, A holds"; furthermore, the intended meaning of $P\mathcal{T}_P A$ is the same as the meaning of $\mathcal{T}_P PA$. We can then formalise the complete two-dimensional well formed formula.

Definition 2.8 *Two-dimensional well formed formula*

The set $\mathcal{W}_{\mathcal{FF}}$ of two-dimensional well formed fomulae is the smallest set such that:

- every propositional letter belongs to it;
- if $A \in \mathcal{W}_{\mathcal{FF}}$ then $\neg A \in \mathcal{W}_{\mathcal{FF}}$;
- if $A \in \mathcal{W}_{\mathcal{FF}}$ and $B \in \mathcal{W}_{\mathcal{FF}}$ then $A \wedge B \in \mathcal{W}_{\mathcal{FF}}$, $A \vee B \in \mathcal{W}_{\mathcal{FF}}$ and $A \rightarrow B \in \mathcal{W}_{\mathcal{FF}}$
- if $A \in \mathcal{W}_{\mathcal{FF}}$ and $B \in \mathcal{W}_{\mathcal{FF}}$ then $S(A, B) \in \mathcal{W}_{\mathcal{FF}}$ and $U(A, B) \in \mathcal{W}_{\mathcal{FF}}$;

- if $A \in \mathcal{W}_{\mathcal{FF}}$ and $B \in \mathcal{W}_{\mathcal{FF}}$ then $\mathcal{T}_S(A,B) \in \mathcal{W}_{\mathcal{FF}}$ and $\mathcal{T}_U(A,B) \in \mathcal{W}_{\mathcal{FF}}$; □

The two-dimensional well formed formulae were constructed by first extending one-dimensional temporal formulae to b-formulae and then to two-dimensional well formed formulae so that $\mathcal{H}_{\mathcal{F}} \subset \mathcal{B}_{\mathcal{F}} \subset \mathcal{W}_{\mathcal{FF}}$. The semantics definition for b-formulae presented in Definition 2.6 can be directly appllied to wff's. The syntactic definitions of the one-place operators as in Definition 2.7 are also valid as abbreviations in well formed formulae.

The idea of a two-dimensional model appears as a generalisation of the one-dimensional model to deal with evolving beliefs. A two-dimensional model can be reduced into a one-dimensional model by means of a projection.

Definition 2.9 *Projection into the historical dimension*

A two-dimensional model $\mathcal{M} = (\mathcal{T}_h, <_h, \mathcal{T}_b, <_b, o', t, g)$ projected into the historical dimension with respect to a reference point o is a one-dimensional model

$$\mathcal{M}_o = (\mathcal{T}_h, <_h, t, g_o)$$

where for all $s \in \mathcal{T}_h$, $g_o(s) = g(o,s)$. □

The projection into the historical dimension gives us the history of the world according to one particular belief. If we project a two-dimensional model into the belief dimension, we finish with a view from the evolution of the beliefs about a particular instant of the history.

Definition 2.10 *Projection into the belief dimention*

A two-dimensional model $\mathcal{M} = (\mathcal{T}_h, <_h, \mathcal{T}_b, <_b, o, t', g)$ projected into the historical dimension with respect to an historical point t is a one-dimensional model

$$\mathcal{M}_t = (\mathcal{T}_b, <_b, o, g_t)$$

where for all $x \in \mathcal{T}_b$, $g_t(s) = g(x,t)$. □

3 Applications to Databases

A database is seen here as a one-dimensional temporal model, called an *historical database* [Snodgrass & Ahn 85], obtained by projecting a two-dimensional model into the historical dimension with respect to the current belief time. A query is an historical formula and query evaluation is the verification of the truth value of a formula in this model. The evaluation is done by the application of rules 1–7 of Definition 2.6 and the substitution rules 1–6 of Definition 2.7.

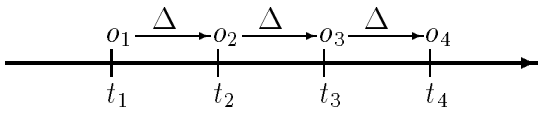
Transactions are logical work units, i.e. logical units of change in databases [Date 86]. A transaction is here associated with a change in the current belief time, which is graphically represented by the symbol $\xrightarrow{\Delta}$. Since the set $\mathcal{T}_b$ is a discrete set,

it can be modelled by the natural numbers; hence, a database in the current belief time o goes, after a transaction, to a current belief time $o+1$ which is represented by $o \xrightarrow{\Delta} o+1$. $\mathcal{T}_h$ is also a discrete set and time is counted in ticks in the historical dimension.

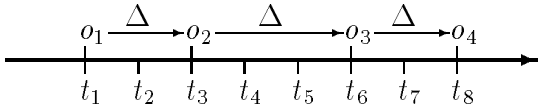
A state of the database is the projection of the two-dimensional model on the historical dimension with respect to some reference time. These definitions of transactions and states are adapted from the definitions given by [Clifford & Warren 83].

Considering some of the possible different relationships between the set of belief time points and the set of historical time points, we are able to describe the synchronism between transactions and ticks as discussed in [TEMPORA 89]:

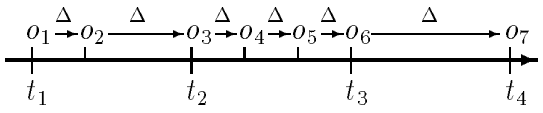
- $\mathcal{T}_h = \mathcal{T}_b$
  This is considered the synchronous case, in which a transaction occurs in the database at every tick. This can represent that every time the observer realises that time is changing, the database is updated. The duration of a state of the database is constant and equal to one tick.



- $\mathcal{T}_b \subseteq \mathcal{T}_h$
  This is considered a middle term between the synchronous and asynchronous cases, wherein transactions do not occur after every unit of time, but transactions are synchronized with the ticks. As a result, states can have a variable duration of an integer number of ticks.



- $\mathcal{T}_h \subseteq \mathcal{T}_b$
  This is one possible asynchronous case in which several transactions may occur between two ticks, but in which there is always a transaction occurring in synchronism with a tick. It can represent that there is a constant update every tick due to, for instance, temporal specification execution, and the transactions introduced between every tick can be caused by the user updating the database at any time.



(Note that in the three cases above, there is no need for two distinct $<$ relations, since it is enough to define it on $\mathcal{T}_h \cup \mathcal{T}_b$, i.e. we have *interlaced flows of time*.)

3.1 Temporal Specifications

According to [Gabbay 89], time can be considered in two ways, namely the declarative view of time and the imperative view of time. Viewing an historical database as a model is considering only the declarative view of time. In order to deal with the imperative view of time, we have to extend the database system to cope with temporal specifications, so that what we will call a *database system* has both a descriptive component (i.e. the model) and an active component (i.e. temporal specifications). A temporal specification, in the database context, is an historical formula of the form:

$$Condition \rightarrow Action$$

Condition is an h-formula, i.e., it is an acceptable query to the database, therefore is can be evaluated according to the rules 1–7 of Definition 2.6. When the evaluation succeeds, the *Action* must be executed.

An action, in principle, could be any historical formula involving only the present and the future (formally, we should say that an action is a boolean combination of pure-present and pure-future formulae, where the notion of a pure formula es defined in [Gabbay 89]) but the execution of an action in the future implies the existence of a strategy to perform this execution, and we certainly want this strategy to guarantee that the action will be executed, i.e., *Action* will eventually hold in a future state of the database. For this discussion, we assume that the flows of time are interlaced in one of the three ways described above.

In order to have a simpler execution strategy, we may make the action language a little less expressive then the original formulation. Let's consider a few problematic cases:

- The execution of an action of the form $A_1 \vee A_2 \vee \ldots \vee A_n$ may cause some problems for the strategy to choose which A_i component to execute in order to execute the action. Since each A_i may demand certain execution at a different historical time, delaying the execution to a future time may make some (eventually all) A_i impossible to execute, because it is impossible to execute an action in the historical past. On the other hand, the immediate execution of some A_i may be impossibe (e.g., the action may require the use of some resources in the environment that are not available at the moment) or undesirable (e.g., the action may be the payment of an invoice that should be delayed as much as possible).

- The execution of an action of the form FA. The problem now is when to execute this action, since its execution may be indefinitely postponed; futhermore, we may have the same problems of impossibility and undesirability as in the last case. We suggest that actions fo the form FA be replaced actions with a two-place operator $A\, atnext\, B$, where $atnext$ is an operator used in [Kröger 87], stating that A must be executed at the next occurence of B in the future. This operator can be defined in terms of the U operator as $A\, atnext\, B \equiv U(A \wedge B, \neg B)$.

By excluding disjunctions and changing FA by $A\,atnext\,B$, we are selecting among the possible actions, only the definite ones. We define then definite action in the following way, wherein a literal is an atomic formula or the negation of an atomic formula:

- every literal is a definite action;
- if A is a definite action, TA is a definite action;
- if A is an action and B is any h-formula, then $A\,atnext\,B$ is a definite action;
- if A_1 and A_2 are definite actions, so is $A_1 \wedge A_2$.

The semantics of definite actions is given by Definition 2.6 with the appropriate transformations $TA \equiv U(A, false)$ and $A\,atnext\,B \equiv U(A \wedge B, \neg B)$.

3.2 Temporal Specifications and Database Updates

A database system is then interacting with the environment it is inserted in. This interaction can happen in two-directions:

- Enviroment $\rightarrow$ Database: the environment (which includes the user) performs an action upon the database, e.g. by updating the database.
- Database $\rightarrow$ Environment: the database triggers the execution of some action in the world, e.g. printing checks, etc.

The interaction database-environment performed in either way is going to cause a change in the database state. A legal update made by the user changes the database state. The execution of a temporal specification will, besides triggering an action upon the environment, cause an update in the database basically because it believes now that an action was triggered. Another way of seeing this update after the execution of an action is by considering it as a reaction of the environment upon the database caused by the executed action.

If A is an atomic formula, its insertion in the database can be described by the two-dimensional well formed formula

$$\mathcal{T}_Y \neg A \wedge A \tag{1}$$

whereas its deletion from the database can be expressed by

$$\mathcal{T}_Y A \wedge \neg A \tag{2}$$

A two-dimensional view of the execution of a temporal specification of the form

$$Condition \rightarrow Action$$

is the expansion of the formula above to

$$(Condition \wedge \neg Action) \rightarrow \mathcal{T}_T Action \tag{3}$$

meaning that if *Condition* holds in one state of the database but not *Action*, then in the next state *Action* is made true. The actual detection of an action to be executed can be described as:

$$\mathcal{T}_Y \neg Condition \wedge (Condition \wedge \neg Action) \tag{4}$$

Had we used formula 4 as the antecedent of formula 3, this would mean that whenever a temporal specification is triggered at a state of the database, it must be executed immediately, so that the temporal specification will hold in the next state of the database; however, this may not be always feasible in practice. Forcing *Action* to hold in the database may also trigger some action in the database environment which cannot immediately be executed, e.g. due to resource allocation. The database system is then actively interfering with the world it is trying to model.

Recall now the retroactive salary rise example, in which an action was executed (namely, the payment of the original salary of \$1000) and then the information on which this action was based was changed (namely, the salary was increased from \$1000 to \$1200). We say that, in this case, the action has become non-supported, which is described by the following formula:

$$\mathcal{T}_P(\,((Condition \wedge \neg Action) \rightarrow \mathcal{T}_T Action)\,) \wedge \neg Condition \tag{5}$$

Note that the subformula under the scope of the $\mathcal{T}_P$ operator is formula 3, meaning that the action was executed in a past state of the database. The second element of the conjunct says that in the current state of the database, the condition that triggered the action is no longer valid.

Still considering the retroactive salary increase example, the state of the database after the increase implies that a different action (namely, the payment of \$1200) shold have taken place in the last payment day. We say that the update has caused an action to be triggered in the past, which is described by the formula:

$$P(\,\mathcal{T}_Y \neg Condition \wedge (Condition \wedge \neg Action)\,) \tag{6}$$

Note that the subformula inside the scope of the P operator is formula 4, which actually means that an action was triggered in the past. This is the first time in which we use a formula that is not a b-formulae, and the reason for this is that in this formula there was an actual need for the evaluation time to be dislocated to the past.

Finally, the case in which a temporal specification is violated must be considered. Since the logical view of the execution of a temporal specification is making *Action* hold in the database, it can happen that, at some state after its execution, the information about *Action* is deleted from the database, therefore violating an already executed specification:

$$\mathcal{T}_P(\,(Condition \wedge \neg Action) \rightarrow \mathcal{T}_T Action\,) \wedge (Condition \wedge \neg Action) \tag{7}$$

The subformula inside the scope of the $\mathcal{T}_P$ operator is formula 3, meaning that the formula was executed in a past state of the database; the rest of the formula specifies that *Condition* still holds in the database but *Action* was deleted.

4 Implementation Aspects

The complete implementation of **dbT**, a prototype of an historical database that supports the execution of temporal specifications is discussed in [Finger 90]. In this paper we will only consider those implementation topics related with the descriptions given by formulae 3 – 7.

The historical and belief flows of time are considered to be interlaced and we have implemented the asynchronous model as described in last section.

Formulae 3 – 7 that were used to describe certain situations in a database are now used as specifications. The implementation performs the following tasks:

- Detection of an action to be executed;
- Execution of an action;
- Detection of non-supported actions;
- Detection of temporal specification violation.

The detection of an action to be executed is performed at the begining of every tick. Therefore, we assume that a tick is a rather large unit of time if compared with "real time", i.e. scanning all the temporal specifications in order to find which has been triggered in the last tick can become a rather expensive task if performed once every second. On the other hand, if it is performed once at the begining of each day, the overhead caused by the process of detecting actions to be executed is acceptable provided the number of temporal specifications is reasonable. What is meant by reasonable is relative, because we can have queries that are checked in a much faster way than others. In this system, since the implementation also checks for new actions to be executed, as well as actions that have become non-supported, after every transaction in the database, a large number of temporal specifications would certainly yield a very inefficient system. The detection of an action is done by querying *Condition* in the database. The query process records also all the instants of time that were used in proving the query; this information is later used for the detection of non-suported actions and we call it *temporal dependencies*.

The execution of actions is a very detailed and complicated matter. It involves the allocation of resources in the database environment, (e.g., we can only print a check if the printer is available), the elaboration of a strategy for the execution of pending actions (e.g., decide what is to be printed first, payment checks or an operation report) and the user interface, among other tasks. Our implementation is only concerned with the effects in the database caused by the execution of an action.

The detection of non-supported actions is done after a transaction occurs in the database. The result of the detection of a non-supported action is a message prompted to the user notifying this fact. The responsibility for taking the appropriate action is left to the user. A future extension of the system should allow for two-dimensional specifications to cope with this situation. Instead of re-evaluating the *Condition* part of all the executed actions, we compare the instants of time associated with the information being changed with the temporal dependencies of each

action. In case of an intersection, the *Condition* part of the action is re-evaluated and in case of failure, the action has become non-supported.

Violation of temporal specification has proved to be a rather normal event in the system, and its believed to be an indicator of the poor design of the specification rules. For instance, still in the payment example, suppose we have the following rule:

$$hire(Person, Amount) \rightarrow G\,salary(Person, Amount)$$

This rule states that whenever a person is hired for an amount of money, his or her salary is going to be this amount. Any subsequent increase in the person's salary will cause the violation of the rule above, since part of the information inside the G operator stating that something "is always going to be" will be deleted. The design of appropriate rules should take into account such situations and try to avoid those violations caused by the simple use of the database.

It must be noted that the detection of actions triggered in the past is not performed by the implementation. The reason is clear. In order to do so, in principle, we should have to search the whole past every time there is an alteration in the database, evaluating all the specifications at all moments of the past. The difficulty can be detected in the description of an action triggered in the past in formula 6, in which the definition of the detection of an action to be executed is completely inside the scope of the P operator, with no clue about what instant in time it has occured. Unless a method is found to select a small set of time points in which the temporal specifications evaluation can be performed instead of scanning the whole past, the detection of actions triggered in the past has a prohibitive computational cost.

5 Summary and Discussions

We have developed in this paper a two-dimensional propositional temporal language in order to simultaneously describe the history of the world and the evolution of beliefs about the world. We have applied this two-dimensional temporal language in the description of historical databases, basically describing the handling of updates and its behaviour due to the incorporation and execution of temporal specification as the active part of a database system. The historical level was described in the fully expressive US-temporal logic, and the second dimension was a generalization of this US-temporal logic to cope with beliefs.

5.1 The Two-dimensional Language Revisited

The two-dimensional temporal language developed so far has the main characteristic of being a propositional language. In order to model a relation (e.g., in order to explicitly model relational databases) the language should be extended to deal with predicates and quantification as well. The notion of two-dimensional interpretations and models should be modified a two-dimensional variable assignment should be introduced. Note that the introduction of quantification over variable also introduces

the problem of which is the domain of quantification, so that the meaning of the following formulae must be made precise:

- $\mathcal{T}_G\, G \forall x\, A(x)$
- $\mathcal{T}_G \forall x\, G\, A(x)$
- $\forall x\, \mathcal{T}_G\, G\, A(x)$

If we assume that the set of elements (domain) we are reasoning about, and therefore quantifying over, is constant, i.e. no new element comes to knowledge or is destroyed in both the historical and the belief flows of time, then the three formulae above are seen as having the same meaning. This restriction may not always reflect the intuition of our evolving beliefs and the evolution of history, but in the database case it seems a reasonable assumption, since it deals with a limited part of the universe in its description that can be supposed constant.

Any of the three formulae above should be read as in all future belief points and in all the future historical points every element of the global domain has the property A. Note that inverting the order of $\mathcal{T}_G$ and G should not alter the meaning of the three formulae above due to the independence between the two dimensions. The constant domain restriction does not imply the equivalence of fomulae like $\exists x\, G\, A(x)$ and $G\, \exists x\, A(x)$.

Even with the constant domain restriction above, there are other problems to be taken into account. For instance, it may be the case or not that the constants of the language are *rigid designators*, i.e. they represent the same object of the domain at all time points and belief points. The assumption of constants being rigid designators may again violate the intuition of changing beliefs, but it is also a reasonable assumption for a database system.

Note as well that the two-dimensional language has no autoepistemic operator that allows it to refer to the current beliefs, like in languages described in [Moore 85] and [Konolige 86]. The two-dimensional laguage has no means to differentiate between what is true from what is believed to be true, since its starting point is that everything that is represented is a belief about the world and there is no way of refering to the world except by the berliefs about it. On the other hand, the main point of the introduction of the second dimension was to provide a way to actually describe the evolution of the beliefs, as opposed to introspection (what an agent believes, and what he believes he believes, and what is true in his beliefs, etc).

Finally, the two-dimensional system presented allows for only one believing entity, as opposed to the system described in [Konolige 86] wherein many different believing entities can coexist and one may have beliefs about what others believe. In this sense, the two-dimensional system has a centralized capability of describing beliefs and their evolution, which suits the desciption fo monolithic database systems but may fail to describe distributed database systems.

5.2 Two Dimensions and Databases Revisited

In the two-dimensional description of databases, the database was seen as a model, namely the projection of a two-dimensional with respect to a belief point; the second dimension ws presented as a witness of the intrinsic temporal nature or databases, since the content of the database is always changing in time. The model theoretic view of databases can now be changed to a proof theoretic view, as in [Lloyd & Topor 85], by giving a two-dimensional description of a deductive database. A resonably efficient inference system must be provided together with proofs of soudness and perhaps completeness in some sense. Another possibility is to provide a means of translating historical formulas to a language in which an inference system already exists, e.g. a Prolog-like language, but some care must be taken for precisely defining the meaning of a temporal negation as failure.

Temporal specifications can play a larger roll in the updating process in the database. As sugested by Peter McBrien, a rule of the form

$$Condition \rightarrow Action$$

can have its *Condition* part evaluated in both the database and the environment, therefore not only the database state would cause temporal specifications to be evaluated, but the actions of the environment towards the database, e.g. the inclusion or deletion of data, would also activate some rule that will be in charge of processing the alteration. This suggests a rule-based database architecture, in which the temporal rules are the central communicating point between the database and its environment, as opposed to a database centred architecture, in which the database communicates directly with its environment and the temporal rules are seen only as an appendix to the database. A further discussion of the consequences of a rule-based database system is discussed in [Finger 90].

Throughout this paper we have been talking about databases without mentioning integrity constraints and the reason is the following. In the model theoretic view of the database, an integrity constraint is any historical formula that must be satisfied by the database in every state. An integrity constraint normaly has the following form, or can be brought to it:

$$\neg BadCondition$$

Since it is not specified what attitude should the database system take in case an update violates some integrity constraint, we suggest that this attitude become part of the repertoir of specifiable actions that the database system can trigger by executing a temporal specification, therefore the above integrity constraint can be seen by the system as a rule of the form:

$$BadCondition \rightarrow Attitude$$

so that the detection of *BadCondition* in the database will trigger an action responsible for the treatment of what should be an integrity constraint violation.

References

[Clifford & Warren 83] J. Clifford and D. S. Warren. *Formal Semantics for Time in Database*, in ACM Transactions on Database Systems, 8(2), 1983, pp. 214–254.

[Date 86] C. J. Date. *An Introduction to Database Systems, volume I*, fourth edition, Addison-Wesley, 1986.

[Finger 90] M. Finger. *A Two Dimensional Approach to Historical Databases*, M.Sc. Thesis, Department of Computing, Imperial College, 1990.

[Gabbay 81] D. M. Gabbay. *Expressive Functional Completeness in Tense Logic*, in Aspects of Philosophical Logic, ed. U. Monnich, Reidel, Dordrecht, 1981, pp. 91–117.

[Gabbay 89] D. M. Gabbay. *The Declarative Past and the Imperative Future* in proceedings, Colloquim on Temporal Logic and Specification, Manchester, April 1987, Ed. B. Banieqbal et al., Springer Lecture Notes in Computer Science 398 (1989).

[Gabbay 90] D. M. Gabbay. *Temporal Logic — Mathematical Foundations and Computational Aspects* [Chapter 7: Basic Many Dimensional Systems], Oxford University Press, to appear.

[Gabbay & Rohrer 78] D. M. Gabbay, C. Rohrer. *Relative Tenses: The interpretation of tense forms which occur in the scope of temporal adverbs or in embedded sentences* in Papers on Tense, Aspect and Verb Classification, TBL Verlag G. Narr, Tubingen, 1978, pp. 99–111.

[Lloyd & Topor 85] J. W. Lloyd, R. W. Topor. *A Basis for Deductive Database Systems*, Journal of Logic Programming, 2:93–109, 1985.

[Konolige 86] K. Konolige. *A Deductive Model of Belief*, Research notes in artificial intelligence, Morgan Kaufmann, 1986.

[Kröger 87] F. Kröger. *Temporal Logics of Programs*, Springer EATCS Monographs on Theoretical Computer Science, Springer-Verlag, 1987.

[Moore 85] R. C. Moore. *Semantical Considerations on Nonmonotonical Logic* in Artificial Intelligence, 25(1), 1985.

[Snodgrass & Ahn 85] R. Snodgrass, I. Ahn. *A taxonomy of Time in Databases*, in Proceedings of ACM SIGMOD International Conference on Management of Data, Ed. S. Navathe, Association for Computer Machinery. Austin, Texas, May 1985, pp. 236–246.

[TEMPORA 89] *Background on Representing Rules in Temporal Logic*, TEMPORA Deliverable T2.1, ESPRIT Project E2469, University of Liege, December 1989.