An extension to EMTPL

María Laura Cobo Marcelo Alejandro Falappa
Department of Computer Science and Engineering *
Universidad Nacional del Sur
Bahía Blanca - Argentina
fax: +54 291 4595136
[mlcobo, mfalappa] @cs.uns.edu.ar

Abstract

The development of languages allowing a proper handling of time is important in many areas of computer science because the capability to deal with the notions of change and time are essential to solve their problems.

In the development of EMTPL [CA99a] we focused the research work in making the languages useful in the area of databases, where some problems demand the possibility to represent and use temporal information. We achieve that goal, although we managed the negation in a way that causes the lost of some properties of the underlying logic. This logic was deeply studied and it is one of the most famous metric temporal logics [Pri67a]. The main idea of this work is to extend EMTPL by the addition of a new operator, and also grant the negation with its logic meaning.

Key words: Programming Languages, Metric Temporal Logic, Deductive Databases, Negation.

1 Introduction

The database research field provides solutions for the problem of storing and retrieving information. One of the possible ways to achieve this goal is by using deductive databases. They use logic as a mean to represent information in a declarative way and an inference procedure to answer queries. Some problems are involved with the notions of change and time. For example we could be interested on keeping record of the different positions of the employees in some factory, evolution of patients in a hospital, transactions in a business or events in an industry environment.

Some implementations are available for languages with capability to handle temporal concepts, for example, *Temporal Prolog* in [CA99b] based on the proposal of [Gab87]. The Temporal Prolog's proposal was improved in [CA99a] where we had presented a programming language, *Metric Temporal Programming Language* (MTPL), with the capability of dealing with more precise temporal information; this language is based on a deeply studied logic proposed by Prior ([Pri67a] and [Pri67b]) and follows one of the most known ways of seeing time, through A-series (past, present and future). There are other logics, and some of them formalize the other view of time, as B-series (before, after), where we use explicit references of time. An example of logic of this later type has been presented by Rescher [RU71]. However, there are many circumstances when it is very useful if not mandatory to have both ways to use and refer time.

To preserve all the properties and benefits of the underlying logic is important to associate the logic meaning to the negation operator. In the first version of EMTPL, we use negation as

*Partially supported by the Secretary of Science and Technology (Universidad Nacional del Sur).

failure instead of classical negation. Because of this the manipulation of negative information is not always appropriate, we could think about the need of counting with an operator that offers to the language some kind of temporal negation. This new operator could be interesting, because we will count with an operator that provides a way to manage that negative information that is vital in some contexts or problems.

EMTPL [CA99a] was designed to manipulate temporal references of both kinds, making the language appropriate for Databases. In section 2 we present a brief presentation of the programming language EMTPL. In section 3 we mention the principal aspects of the extension we have in mind. Conclusions and future work suggestions will be presented in section 4.

2 Brief revision to EMTPL

2.1 Syntax and semantics

The syntax of the EMTPL's operators as follows:

$$\phi = p|\neg\phi|\phi_1 \wedge \phi_2|AT_n\phi|\diamondsuit_n\ \phi|\exists n\ \diamondsuit_n\ \phi|\forall n\ \diamondsuit_n\ \phi|\diamondsuit_n\ \phi|\exists n\ \diamondsuit_n\ \phi|\forall n\ \diamondsuit_n\ \phi$$

Disjunction and implication could be defined as usual. More temporal operators are definable as specified below.

The intuitive meaning of the temporal operators $\diamondsuit$ and $\diamondsuit$ is as follows:

- $\diamondsuit\ \phi$ sometime in the future ϕ or ϕ will be true in some moment of the future.
- $\diamondsuit\ \phi$ sometime in the past ϕ or ϕ was true in some moment of the past.

With this intuitive idea we can now give the formal semantic the operators. We assume, then, a set of instants $\ldots, i_n, i_{n+1}, i_{n+2}, \ldots$ that represents an unbounded and linear temporal structure. The information in the system could be seen as a list of sets, σ, that contains all the truths we have in our theory. Each member of the list is a set denoted by $\sigma(j)$. Then $\sigma(j)$ represents the set of truths at an instant j. We follow with an inductive definition for "a temporal proposition true at an instant j". We start specifying that $p \in \sigma(j)$ means "p is true at instant j".

It is important to point out that the quantification is applied only over the members of the temporal structure, *i.e.*, the set of numbers we choose to represent time. We claim this set of numbers to be a metric space.

$(\sigma, j) \models p$ iff $p \in \sigma(j)$
$(\sigma, j) \models \neg p$ iff $(\sigma, j) \not\models p$
$(\sigma, j) \models p \wedge q$ iff $(\sigma, j) \models p$ and $(\sigma, j) \models q$
$(\sigma, j) \models \diamondsuit_c\ p$ iff $(\sigma, j + c) \models p$
$(\sigma, j) \models \diamondsuit_c\ p$ iff $(\sigma, j - c) \models p$
$(\sigma, j) \models \exists n\ \diamondsuit_n\ p$ iff exists an instant $n,\ n > 0$ such that $(\sigma, j + n) \models p$
$(\sigma, j) \models \exists n\ \diamondsuit_n\ p$ iff exists an instant $n,\ n > 0$ such that $(\sigma, j - n) \models p$
$(\sigma, j) \models \forall n\ \boxplus_n\ p$ iff for all instant $n,\ n > 0$ we have $(\sigma, j + n) \models p$
$(\sigma, j) \models \forall n\ \boxminus_n\ p$ iff for all instant $n,\ n > 0$ we have $(\sigma, j - n) \models p$
$(\sigma, j) \models AT_c\ p$ iff $(\sigma, c) \models p$

We can define some other nice operators from this ones [CA00, CA99a]

2.2 Definition of the language

The language is based on Prior's logic [Pri67a, Pri67b] plus an extension to that logic buy the introduction of the AT. This extension consists mainly, on introducing the AT operator in Prior's axiomatization [CA99a].

In what respects to the language based on this extended logic, *Extended Metric Temporal Programming Language* (EMTPL), we define it in this way:

Definition **1** (EMTPL)
Let us consider a language with propositional atoms, the logic connectives: $\wedge$,$\vee$ $\rightarrow$, $\neg$ and the temporal operators: $\diamondsuit_n$, $\diamond_n$, $\diamondsuit$[1], $\diamond$, $\boxplus$[2], $\boxminus$. We define the notion of EMTP's *program, clause, always clause, ordinary clause, head, body* and *goal* as follows:

- A *program* is a set of "clauses".
- A *clause* is AT_iC or C where C is an "ordinary clause".
- An *ordinary clause* is a "head" or a $B \rightarrow H$, where B is a "body" and H is a "head".
- A *head* is an atomic formula, $\diamondsuit_n\ A$ or $\diamond_n\ A$ or $\boxplus A$ or $\boxminus A$ where A is a conjunction of "ordinary clauses".
- A *body* is an atomic formula, a conjunction of bodies, $\diamondsuit_n\ B$ or $\diamond_n\ B$ or $\boxplus A$ or $\boxminus A$ or $\neg B$ where B is a body.
- A *goal* is AT_nB or B where B is any body or a disjunction of bodies.

■

It is important to point out that the facts or rules ($Prop$) that are not affected with an AT_n operator are interpreted as facts or rules that happen in the present moment, *i.e.*, we are going to put them in the base as AT_mProp, where m means "now". The reader can find more details of this language in [CA99a].

3 Motivation for a negation extension

The definition of EMTPL includes the operator $\neg$, this operator can be part of a clause, more specifically it can form part of the body of an EMTPL clause. We have previously seen that the semantic of $\neg$ operator is:

$$(\sigma, j) \models \neg p \text{ iff } (\sigma, j) \not\models p$$

Intuitively this is the negation as failure semantic, because it means that some formula $\neg A$ is **true** if and only if the system can not prove the truth of A. This semantic is quite different from the classical negation semantic, in which the truth of $\neg A$ depends only in the proof of the negated formula. The decision of which semantic choose is quite important, because no matter which one the language support, it causes some effects on it. If we choose negation as failure, the system looses some properties of the underlying logic and a considerable set of theorems. Another disadvantage of this way of conceive negation is the restriction to manipulate negative information. That means that we can not have in the base any negative information, this can be seen in the fact that the negation operator can be only part of a body clause and that body must be proven in the system. On the other hand since databases are usually systems with incomplete information, classical negation will not always has an positive or negative answer

[1] $\diamondsuit\phi =_{Def} \exists n\ \diamondsuit_n\ \phi$
[2] $\boxplus\phi =_{Def} \forall n\ \diamondsuit_n\ \phi$

to all possible queries. Instead the system could answer **true**, **false** and **unknown**. In the development of EMTPL we though of critical data bases where the answer **unknown** is quite dangerous.

Considering this we can think in the possibility of adding a new negation operator to EMTPL language, *i.e.* the language keep having negation as failure, but provides some kind of temporal negation, that allows manage negative information.

3.1 Extension 1: $EMTPL_{\nabla}$

$EMTPL_{\nabla}$ is like EMTPL, except for the new operator ∇. The syntax of the new language $EMTPL_{\nabla}$ is as follows:

$$\phi = p|\neg\phi|\phi_1 \wedge \phi_2|AT_n\phi|\nabla\phi|\Diamond\!\!\!\!+_n\ \phi|\exists n\ \Diamond\!\!\!\!+_n\ \phi|\forall n\ \Diamond\!\!\!\!+_n\ \phi|\Diamond_n\ \phi|\exists n\ \Diamond_n\ \phi|\forall n\ \Diamond_n\ \phi$$

Assuming a temporal structure like EMTPL's one, the semantic of the new operator is:

$(\sigma, j) \models \nabla p$ iff $\nabla p \in \sigma(j)$ or $(\sigma, j) \models \neg p$

The intuitive meaning of this semantic is: "∇p is true only if we have ∇p in the set of proven facts, or we can not prove p" In order to avoid the *or-problems* of logic programming we only are going to allow the negation of atomic formulas.

As a consequence of the definition we have the following definition of $EMTPL_{\nabla}$

Definition **2** ($EMTPL_{\nabla}$)
Let us consider a language with propositional atoms, the logic connectives: $\wedge$,$\vee$ $\rightarrow$, $\neg$, ∇ and the temporal operators: $\Diamond\!\!\!\!+_n$, $\Diamond_n$, $\Diamond\!\!\!\!+$, $\Diamond$, $\boxplus$, $\boxminus$. We define the notion of $EMTP_{\nabla}$'s *program, clause, always clause, ordinary clause, head, body* and *goal* as follows:

- A *program* is a set of "clauses".
- A *clause* is AT_iC or C where C is an "ordinary clause".
- An *ordinary clause* is a "head" or a $B \rightarrow H$, where B is a "body" and H is a "head".
- An *atomic formula* is an atom or ∇A where A is an atom.
- A *head* is an atomic formula, $\Diamond\!\!\!\!+_n\ A$ or $\Diamond_n\ A$ or $\boxplus A$ or $\boxminus A$ where A is a conjunction of "ordinary clauses".
- A *body* is an atomic formula, a conjunction of bodies, $\Diamond\!\!\!\!+_n\ B$ or $\Diamond_n\ B$ or $\boxplus A$ or $\boxminus A$ or $\neg B$ where B is a body.
- A *goal* is AT_nB or B where B is any body or a disjunction of bodies.

■

For example we can now infer negative information like in the following rule:

$$in\_way(Car, Street) \wedge move(Car) \ \rightarrow\ \nabla free\_way(Street)$$

In EMTPL, the previous rule is not allowed because it is meaningless to infer something whose value depends on the failure of some other proof. Also, we can consider a short example where the use of ∇ is different form $\neg$.

$$\textbf{op}\ move(Car) \wedge in\_way(Car, Street2) \wedge lateral\_way(Street, Street2) \rightarrow\ free\_way(Street)$$

In the previous rule if there are negative information the meaning change if we replace **op** with $\neg$ or with ∇. That is because it is not the same infer that a car is not moving because we just know that the car is not moving.

3.2 Extension 2: $EMTPL_{\nabla_c}$

The new operator could be used as the realization operand AT or like $\Diamond_c$ but for negative information. In this sense we can analyze what happens with an operator of this kind.

The syntax of the language is quite similar to the corresponding to $EMTPL_{\nabla}$ the only modification is that we replace ∇p by $\nabla_c\ p$

Let see the semantic of ∇_c, again we assume a temporal structure like EMTPL's one.

$$(\sigma, j) \models \nabla_c\ p \text{ iff } (\sigma, j + c) \models \neg p$$

With this semantic we do not manage negative information in the same way as in 3.1, because the manipulation of that kind of information is, in some way, virtual.

We can think in having both operators, but it seems to be not very attractive because al first sight we can think that:

$$\nabla_c\ p = \Diamond_c \nabla p$$

Although it can be interesting to analyze what happen with this kind of operators more carefully.

4 Conclusions and future work

We show some ways of extend the language EMTPL to manage negative information. The idea behind the first one is to combine classical negation and negation as failure, so we can have negative information and also keep a two-valuate system. For this alternative we believe that can be interesting to find a way to make the operator ∇ more flexible, *i.e.*, analyze what happens with the system if we extend its application to any formula instead of only atomic ones The idea of the second extension is to provide some kind of temporal negation, this can lead to some interesting operator.

In both cases it should be analyzed how this extensions affects the expressiveness of the language, and at the same take a look to the underlying logic part. Another aspect to consider is how to introduce these operators in an interpreter of our language.

References

[CA99a] María Laura Cobo and Juan Carlos Augusto. EMTPL: A Pogramming Language for Temporal Deductive Data Bases. In *Proceedings de la* XIX International Conference of the Chilean Computer Science Society, pages 170–178, Talca, Chile, 1999.

[CA99b] María Laura Cobo and Juan Carlos Augusto. Fundamentos lógicos e implementación de una extensión a temporal prolog. *The Journal of Computer Science and Technology (JCS&T), sponsored by ISTEC (Iberoamerican Science & Technology Education Consortium*, 1(2):22–36, 1999.

[CA00] María Laura Cobo and Juan Carlos Augusto. Towards a Programming Language Based on Prior's Metric Temporal Operators. In *Proceedings del VI Congreso Argentino de Cs. de la Computación, CACiC2000*, pages 453 –464, 2000.

[Gab87] Dov Gabbay. Modal and temporal logic programming. In Antony Galton, editor, *Temporal Logic and their Applications*, pages 197–236. Academic Press, 1987.

[Pri67a] Arthur Prior. *Past, Present and Future.* Clarendon Press, 1967.

[Pri67b] Arthur Prior. Stratified metric tense logic. *Theoria 33*, pages 28–38, 1967.

[RU71] Nicholas Rescher and Alasdair Urquhart. *Temporal Logic.* Springer-Verlag, 1971.