Towards a Programming Language Based on Prior’s Metric Temporal Operators

María Laura Cobo           Juan Carlos Augusto
G.I.S.Di, ¹ - I.C.I.C. ²
Department of Computer Science ³
Universidad Nacional del Sur
Bahía Blanca - Argentina
[mlcobo, jca] @cs.uns.edu.ar

Key Words: Programming Languages, Databases, Real-Time Systems.

Abstract

Many different areas in Computer Science - like Real Time Systems, Temporal Data Bases, Program Verification and Multimedia - demand an adequate way to represent and use the notions of time and change. We present a programming language which is defined starting from a metric temporal logic previously considered by Prior and Rescher ([Pri67a], [Res66]).

The language is based on the declarative approach and combines temporal operators to represent order notions with the capability to refer to distance notions. These features provide a very useful tool to handle practical situations with a formal and clear theoretical basement.

We provide an algorithm for the implementation of an interpreter of this language, based on the notion of a labelled computation tree [Gab87]. It can be shown that all queries lead to finite computations. Some examples are included to illustrate its behaviour.

¹Dynamic Systems Research Group.
²Institute of Computer Science and Engineering.
³Partially supported for the Secretary of Science and Technology (Universidad Nacional del Sur).
1 Introduction

Many different areas in Computer Science demand an adequate way to represent and use the notions of time and change. By mentioning a few of them we can consider Real Time Systems, Temporal Data Bases, Program Verification and Multimedia. As a result some temporal languages have been proposed as a way to handle time in computational systems ([Gab87], [BCW93], [CA98] and [CA99a]). We will focus this work on the declarative paradigm. Our aim is to bring a language which explores a combination of nice features from the main existing approaches.

Researchers in the field have usually found inspiration in already existing temporal logics which can be split in two main groups: those obtained starting from a first-order logic [RU71] and those obtained by adding temporal operators to a propositional logic [Pri67a]. Languages using some kind of first order logic usually have easier ways to handle calendar-related temporal references since an argument in each predicate can be considered an explicit temporal reference to a date. In the other approach, temporal references are closer to natural language avoiding quantification and referring time through the temporal operators in an implicit way.

In this work we will try to get a declarative language based on a metric temporal logic preserving a nice way to express temporal references and adding the possibility to do subtler ones, mostly regarding the notion of duration and the possibility to point out specific time points. These are important features which can be vital for an adequate handling of temporal phenomena in practical contexts. For example when we need to post integrity constraints in a temporal database like: “a person is considered an employee, since he is contracted until he is fired or he resigns” or in a real-time system: “If some time in the future the temperature of a tank increases beyond a safety value then a valve must open”. In the next section we describe the logical layer which is used as a basis for the temporal programming language and the language, MTPL, itself. It is also shown that we can get full first-order expressivity in our logical framework. After that, we define the programming language MTPL and an algorithm for an interpreter of this language. We explain how queries are handled and why they can always be expected to provide an answer, even when they refer to potentially infinite notions like “sometimes in the future” or “always in the future”. The proposal is supplemented with examples illustrating how the algorithm works and the capabilities of MTPL to capture useful situations.

2 The logical layer

We will describe in this section the logical foundations of the programming language to be offered later. The logical side of our proposal is borrowed from Prior’s pioneering work in metric temporal logics ([Pri67a] and [Pri67b]) which provides us a well-known and safe departure point for our framework.

Prior’s metric temporal logic allows us to refer to specific moments of time, improving in that way the subtlety of the temporal language regarding his own non-metric operators \( \Diamond \) (sometimes in the future) and \( \lozenge \) (sometimes in the past). As a result of this departure point we get more expressivity in our language than in other temporal programming languages like that offered in [Gab87] and a clearer way to express useful situations than that considered in [CA99a]. These facts will become better understood after considering some examples at the end of this work.

This logic could be briefly described as a propositional logic augmented with temporal oper-
ators and restricted quantification over variables representing amounts of instants. For example, $\exists n \otimes_n \phi$ means “it is the case $n$ instants ahead that $\phi$ is true” where the term “ahead” must be read relative to “the present” moment, when the sentence is being read.

We can take $n$ in any set of non-negative numbers, the choice depends on what is appropriate for the problem of interest. Some possibilities are the sets of integers, naturals, reals or rationals numbers. In this work we will assume a set of non-negative numbers to be described in further detail in the next section.

2.1 The Logical Language

The temporal structure, $T$, is a set of instants $\ldots, i_n, i_{n+1}, i_{n+2}, \ldots$, ordered by the relation $\leq$. The structure represents a discrete, linear and unbounded conception of time. The syntax of the language for the metric temporal logic to be considered in this work is as follows:

**Definition 1** Let $c$ to denote a constant, $c \in T$, and $n$ to denote a variable with interpretation in $T$. If $\phi$ is an atomic proposition, then the language $\mathcal{M}$ is defined as the following set of well-formed formulas: $\phi = p | \neg \phi | \phi_1 \wedge \phi_2 | \otimes_n \phi | \otimes_c \phi | \exists n \otimes_n \phi | \exists n \otimes_c \phi$

We assume the usual definitions and abbreviations on languages of this sort to consider disjunction and implication as well as:

$$\forall n \otimes_n \phi =_{def} \neg \exists n \otimes_n \neg \phi$$

The information in the system could be seen as a list of sets, $\sigma$, that contains all the truths we have in our theory. Each member of the list is a set denoted by $\sigma(t)$. Then $\sigma(t)$ represents the set of truths at an instant $t$. We consider now what we mean by “a temporal proposition $\phi$ to be true at an instant $t$”, abbreviated as $(\sigma(t) \models \phi)$:

$$(\sigma, t) \models \phi \text{ iff } \phi \in \sigma(t) \text{ where } \phi \text{ is an atomic proposition}$$

$$(\sigma, t) \models \neg \phi \text{ iff } (\sigma, t) \not\models \phi$$

$$(\sigma, t) \models \phi_1 \wedge \phi_2 \text{ iff } (\sigma, t) \models \phi_1 \text{ and } (\sigma, t) \models \phi_2$$

$$(\sigma, t) \models \otimes_c \phi \text{ iff } (\sigma, t + c) \models \phi \text{ where } c \text{ is a constant}, c \in T$$

$$(\sigma, t) \models \otimes_n \phi \text{ iff } (\sigma, t - c) \models \phi \text{ where } c \text{ is a constant}, c \in T$$

$$(\sigma, t) \models \exists n \otimes_n \phi \text{ iff there exists an instant } n, n > 0, \text{ such that } (\sigma, t + n) \models \phi$$

$$(\sigma, t) \models \exists n \otimes_n \phi \text{ iff there exists an instant } n, n > 0, \text{ such that } (\sigma, t - n) \models \phi$$

In case that $\forall n \otimes_n \phi$ and $\forall n \otimes_n \phi$ is added to the basic language then the semantic for them is:

$$(\sigma, t) \models \forall n \otimes_n \phi \text{ iff for all instant } n, n > 0, \text{ we have } (\sigma, t + n) \models \phi$$

$$(\sigma, t) \models \forall n \otimes_n \phi \text{ iff for all instant } n, n > 0, \text{ we have } (\sigma, t - n) \models \phi$$

It is important to point out that quantification is applied only to members of the temporal structure. We will assume that a formula without a temporal operator in its outest level is an abbreviation for $\otimes_0 \phi = \otimes_0 \phi$ and it is interpreted as “$\phi$ is true now”. Our framework is basically instant-based but we show how to add interval-based temporal references in the next section.
2.2 Some Remarks on Expressivity

Usually, operators-based temporal languages resort to the now classical F (“sometime in the future”), P (“sometime in the past”), G (“always in the future”) and H (“always in the past”) which we write in this work as $\Diamond$, $\ominus$, $\Box$ and $\square$ respectively. All these operators are not immediately available either for the programmer or the user in the language we defined above but it is important to see that all of them can be obtained from the framework defined above. Prior himself [Pri67a] have shown that we can obtain the non-metric operators from the metric ones as follows:

\[
\begin{align*}
\Diamond & =_{\text{def}} \exists n \ominus n \phi \\
\Box & =_{\text{def}} \forall n \ominus n \phi
\end{align*}
\]

Definitions for “Next”, $\oplus$, and “Previous”, $\ominus$, [CA98] can also be obtained straightforwardly:

\[
\begin{align*}
\oplus & =_{\text{def}} \assertion{1} \Diamond \\
\ominus & =_{\text{def}} \Diamond \assertion{1}
\end{align*}
\]

According to our previous definitions and since we are considering discrete time the operator $\ominus_n \phi$ can be read in this work as $\ominus_n \ldots \ominus \phi$. Analogously for $\ominus_n \phi$ and $\ominus$. The reader interested in a language having the non-metric operators as its starting point is referred to [Gab87], [CA98] and [CA99a]. Another issue that could not be obvious for the reader is the capability of this language to represent non-instantaneous temporal references. This allows us to consider events whose occurrence lasts during a period of time. We show how we can refer to period-related assertions through the following definitions which introduce two operators similar to $\ominus_n$, and $\ominus_n$, but replacing the number $n$ for an interval:

\[
\begin{align*}
\ominus_{[0,1]} \phi & =_{\text{def}} \phi \lor \assertion{1} \phi \\
\ominus_{[0,n]} \phi & =_{\text{def}} \phi \lor \assertion{1} (\ominus_{[0,n-1]} \phi)
\end{align*}
\]

The definition of $\ominus_{[i_1,i_2]} \phi$ for $i_1 \neq 0$ is easily obtained as follows: $\ominus_{[i_1,i_2]} \phi =_{\text{def}} \assertion{i_1} (\ominus_{[i_2-i_1]} \phi)$. We can obtain the appropriate definition for $\ominus_{[0,n]}$ and $\ominus_{[i_1,i_2]}$ applying the mirror image rule \footnote{The mirror image rule asks to change all past operators by the corresponding future operators and vice versa.} to the previous definitions. A definition for $\ominus_{[i_1,i_2]} \phi$ can be obtained straightforwardly following the same steps we use above. The combination of $\ominus$ for $\ominus$ with $\ominus$, $\Box$, and $\square$ [CA99a] leads to an interesting level of expressivity which could also be recreated using nowadays well-known Kamp’s Since and Until operators, where: $\phi_1$ Since $\phi_2$ means “$\phi_1$ has been true since $\phi_2$ was” and $\phi_1$ Until $\phi_2$ means “$\phi_1$ will be true until $\phi_2$ is”.

This version of Since and Until is called “strong” because it takes for granted the existence of an instant where $\phi_2$ is true. It is also said to be “reflexive” because $\phi_1$ must be true in the instant when “$\phi_1$ Since $\phi_2$” or “$\phi_1$ Until $\phi_2$” is evaluated. The importance of these operators comes from the following results given in [Kam68] and [GHR94]:

**Theorem 1** Since and Until are complete over discrete linear structures with respect to first-order expressively notions.

**Theorem 2** Since and Until can define Prior’s $\Diamond$, $\ominus$ but the converse is not true.

Then having Since and Until in a temporal language or an equivalent expressivity is important because it leads to the greatest expressivity in terms of first order concepts. Both operators can be rewritten using $\oplus$ and boolean connectives in the following way:

\[
\begin{align*}
\phi_1 \text{ Since } \phi_2 & =_{\text{def}} \phi_2 \lor (\phi_1 \land \ominus \phi_1 \text{ Since } \phi_2) \\
\phi_1 \text{ Until } \phi_2 & =_{\text{def}} \phi_2 \lor (\phi_1 \land \ominus \phi_1 \text{ Until } \phi_2)
\end{align*}
\]
The interesting fact from the perspective of this work is that we can show that it is possible to equate that expressivity with our language:

**Theorem 3** The language obtained by adding *Since* and *Until* to a propositional language and $M$ have the same expressivity. 

**Proof 1** It can be seen that *Since* and *Until* can be expressed in $M$:

$$
\phi_1 \text{ Since } \phi_2 =_{\text{def}} \exists n \Diamond n \phi_2 \wedge \Diamond[0,n-1] \phi_1
\quad
\phi_1 \text{ Until } \phi_2 =_{\text{def}} \exists n \Diamond n \phi_2 \wedge \Diamond[0,n-1] \phi_1
$$

The converse is also true, which could be shown by the usual way [GHR94]. Let us use $\text{True}$ as an abbreviation of a tautology in a given instant and $\text{False}$ for $\neg \text{True}$, then:

$$
\Diamond \phi =_{\text{def}} \text{ True Since } \phi
\quad
\Diamond \phi =_{\text{def}} \text{ True Until } \phi
\quad
\Diamond \phi =_{\text{def}} \text{ False Since } \phi
\quad
\Diamond \phi =_{\text{def}} \text{ False Until } \phi
$$

The above definitions show how the basic language $M$ is able to deal with many kinds of temporal references considering a wide range of precision and application. The set of operators considered in the last section can be incorporated to our proposal in many ways. To avoid expressive redundancy in our proposal we choose to provide translation schemas that could be added to our system converting queries with these operators to purely $M$-based queries. Other options which allow programs to have facts and rules using these operators are left for the implementation level. See [CA98] and [CA99a] for some discussions about that.

### 2.3 The metric temporal logic

Now we are going to present the logic, used as a base for the algorithm that can be seen in section 3. A similar logic was presented by Prior in [Pri67a] and [Pri67b].

**Definition 2** The metric temporal logic, has the following set of axioms and rules of inference.

1. All theorems and rules of the propositional calculus plus a theory of quantification over the metric references of the temporal operators.

2. The following axiom schemas (Ax6...Ax8 imply the use of a transitive, linear and unbounded temporal structure)

   - **Ax1**: $\Diamond_n (p \rightarrow q) \rightarrow (\Diamond_n p \rightarrow \Diamond_n q)$
   - **Ax2**: $\Diamond_n \Diamond_n p \rightarrow p$
   - **Ax3**: $\Diamond_m \exists n \Diamond_n p \rightarrow \exists n \Diamond_m \Diamond_n p$
   - **Ax4**: $\Diamond_m \exists n \Diamond_n p \rightarrow \exists n \Diamond_m \Diamond_n p$
   - **Ax5**: $\Diamond_{(m+n)} p \rightarrow \Diamond_m \Diamond_n p$
   - **Ax6**: $\Diamond_m \Diamond_n p \rightarrow (\Diamond_{(m+n)} p)$
   - **Ax7**: $\Diamond_n \neg p \rightarrow \neg \Diamond_n p$
   - **Ax8**: $\Diamond_n p \vee \Diamond_n \neg p$

   and those obtained by applying the mirror image rule.

3. The inference rules: if $\vdash \alpha$ then $\vdash \Diamond_n \alpha$ ; if $\vdash \alpha$ then $\vdash \Diamond_n \alpha$
Rules and axioms Ax1, Ax1’, Ax2 and Ax2’ were rewritten in a simpler but equivalent way [Gar79] regarding those given in [Pri67b]. Conversees of Ax3, Ax3’, Ax4 and Ax4’ can be deduced from the logic [Pri67b] showing that existential quantification is transparent to constant references. Based on this fact we will omit existential quantification in the programming language, using $\Diamond$, $\Box$, $\square$ and $\Box$ instead of the equivalent quantified formulas as defined at the beginning of section 2.2. Another interesting matter mentioned by Prior in [Pri67a] is the possibility of adding the following axioms

$$\Diamond_m \Diamond_n p \rightarrow \Diamond_{m-n} p \quad \text{for } m \geq n \quad \text{and} \quad \Diamond_m \Diamond_n p \rightarrow \Diamond_{m-n} p \quad \text{for } m \leq n$$

Prior does not use them because it is pretty uncomfortable to use conditional axioms from a logical point of view, but they can be very practical and worth considering at the implementation level.

## 3 The programming language

The main purpose of this section is to define a new language for programming in temporal logic. This language considers Prior’s metric operators providing the possibility to refer to specific time points. However we keep the natural advantage of using modal operators.

The following definition is inspired by the one given by Gabbay [Gab87] for classical, non metric, operators. It is important to point out that our language is a subset of the well-formed formulas of $\mathcal{M}$. We choose that subset guided by efficiency issues, much in the same way that Gabbay did with Temporal Prolog.

**Definition 3** Let us consider a language, $\text{MTPL}$, with propositional atoms, the logic connectives: $\land, \lor, \to, \neg$ and the temporal operators: $\Diamond_n$, $\Box_n$, $\Diamond$, $\Box$ and $\square$. A *program* is a set of “clauses”, where a clause can be either an “always clause” ($AC$) or an “ordinary clause” ($OC$). An $AC$ is $\Box A$ or $\Box A$, where $A$ is an $OC$.

An $OC$ is a “head” or a $B \to H$, where $B$ is a “body” and $H$ is a “head”.

A *head* is an atomic formula, $\Diamond_n A$, $\Box_n A$, $\Diamond A$ or $\Box A$ where $A$ is a conjunction of $OC$.

A *body* is an atomic formula, $\bigwedge_i B_i$, $\Diamond_n B$, $\Box_n B$, $\Diamond A$, $\Box A$ or $\neg B$ whith $B$ a body.

A *goal* is any “body” or a disjunction of “bodies”.

We try to avoid the problems involved in the use of disjunctions, allowing their use in a restricted way, i.e., to give more flexibility to the user to express some queries and also for an internal use in the language. The algorithm that we offer next can be used in the implementation of an interpreter for this language. It is based on the notion of *labelled computation tree* proposed by Gabbay [Gab87] but it has important changes to adapt it to our metric language.

**Definition 4** (adapted from [Gab87]) Let $P$ be a finite program and $G$ a goal. We define the notion of a *labelled computation tree* for the success or the finite failure of $G$ from $P$. If $G$ succeeds we write $P?G=1$ and if $G$ finitely fails we write $P?G=0$.

Let $(T, \leq, 0, V)$ be a *labelled computation tree*, such that $0 \leq t$ for all $t \in T$. $V$ is a labeling function giving each $t \in T$ a triple $(P(t), G(t), X(t))$, where $P(t)$ is the program in the moment $t$, $G(t)$ is the goal in the moment $t$ and $X(t)$ is the purpose for that goal. If $X(t) = 0$ the purpose is that the goal be responded in a negative way, if $X(t) = 1$, the purpose is that the computation succeeds.
An important observation is that $P(j)$ plays the role of $\sigma(j)$ in the semantics. $P$ will always be finite, then so will be $P(j)$ for each $j$, see lemma 1 in next section. The algorithm we present below has as input a goal, that we call $G$, a purpose for that goal, that is represented by $X$ and a program in metric temporal logic, $P$. The output of the algorithm is the answer that the query succeeds or fails with the purpose $X$ given $P$. 

**Algorithm** (adapted from [Gab87]) $(T, \leq, 0, V)$ is a labelled computation tree for $P \models G = x$ if and only if the following conditions are satisfied:

1. $V(0) = (P, G, x)$

2. If $t \in T$ is an endpoint and $X(t) = 1$ then $G(t)$ is an atom $q$ and $q \in P(t)$ or $\Diamond_0 q \in P(t)$

3. If $t$ is an endpoint and $X(t) = 0$ then
   
   (a) $G(t)$ is an atom $q$ and $q$ is not the head of an ordinary clause.
   
   (b) $G(t)$ has the form $\Diamond_n A$ (or $\Diamond_{n} A$) and there are no clauses with heads of the form $\Diamond_n D$ (respectively $\Diamond_n D$).

4. If $t$ is not an endpoint, $X(t) = 1$ and $G(t)$ is an atom $q$ then $t$ has exactly one immediate successor $s$ in the tree with $P(s) = P(t), X(s) = 1$ and $G(s) \rightarrow q \in P(t)$

5. If $t$ is not an endpoint, $X(t) = 0$ and $G(t)$ is an atom $q$, then $t$ has $s_1, \ldots, s_n$ as immediate successors in the tree with $P(s_i) = P(t), X(s_i) = 0$, and for some $0 \leq k \leq n$ the clauses $G(s_i) \rightarrow q$ for $i \leq k$ are exactly all the clauses of the above form in $P(t)$

6. If $t$ is not an endpoint, $X(t) = 1$ and $G(t) = A_1 \land A_2$, then $t$ has exactly two immediate successors $s_1$ and $s_2$ in the tree with $P(s_i) = P(t), X(s_i) = 1$ and $G(s_i) = A_i$

7. If $t$ is not an endpoint, $X(t) = 0$ and $G(t) = A_1 \land A_2$, then $t$ has exactly one immediate successor $s$, $P(s) = P(t), X(s) = 0$ and for some $i \in 1, 2$ $G(s) = A_i$

8. If $t$ is not an endpoint and $G(t) = \neg A$, then $t$ has exactly one immediate successor $s$, and $P(s) = P(t), X(s) = 1 - X(t)$ and $G(s) = A$

9. If $t$ is not an endpoint, $X(t) = 1$ and $G(t) = \Diamond_n A$, then $t$ has exactly one immediate successor, $s$, and the next condition holds: $X(s) = 1, G(s) = A$ and

   $$P(s) = \{ C | \Diamond_n C \in P(t) \} \cup \{ C | \Box C \in P(t) \} \cup \{ \Box C | \Box C \in P(t) \} \cup \{ \Diamond_n C | \Box C \in P(t) \} \cup \{ \Diamond_n C | C \in P(t) \text{ and } C \text{ is an } OC \} \cup \{ \Diamond_{(m-n)} C | \Diamond_m C \in P(t) \text{ and } n > m \} \cup \{ \Diamond_{(m+n)} C | \Diamond_m C \in P(t) \} \cup \{ \Diamond_{(m-n)} C | \Diamond_m C \in P(t) \text{ and } m \geq n \}$$

or it has two successors, $s_1$ and $s_2$, and one of the following conditions holds:

   (a) $P(s_1) = P(t)$ and $X(s_1) = 1$ and for some $H$

      i. $G(s_1) \rightarrow \Diamond_p H \in P(t)$,

      ii. $P(s_2) = P(s) \cup \{ H \}$, where $P(s)$ is the set specified in the item above replacing all the appearances of $n$ for $p$.

      iii. $X(s_2) = 1$ and $G(s_2) = \Diamond_{n-p} A$ if $n \geq p$ or $\Diamond_{p-n} A$ if $p > n$. 

(b) idem to item 9a replacing \( G(s_1) \rightarrow \Diamond_p H \in P(t) \) by \( G(s_1) \rightarrow \Diamond_p H \in P(t) \), we apply the mirror image rule to \( P(s_2) \) and \( G(s_2) = \Diamond_{n+p} A \).

(c) \( P(s_1) = P(t) \) and \( X(s_1) = 1 \) and for some formula \( \Box(B \rightarrow \Diamond_p H) \in P(t) \):

\[
G(s_1) = \Diamond_q B, \ X(s_2) = 1, \ G(s_2) = \Diamond_{n-(p+q)} A \text{ if } n > p + q \text{ or } \Diamond_{(p+q)-n} A \text{ if } p + q \geq n
\]

and

\[
P(s_2) = \{ C|\Diamond_{p+q} C \in P(t) \} \cup \{ C|\Box C \in P(t) \} \cup \{ H \} \cup \{ \Box C|C \in P(t) \} \cup \\
\{ \Diamond_{p+q} \Box C|C \in P(t) \} \cup \{ \Diamond_{p+q} C|C \in P(t) \text{ and } C \text{ is an OC } \} \cup \\
\{ \Diamond_{(p+q)-(t+m)} C|\Diamond_m C \in P(t) \text{ and } p+q > t+m \} \cup \\
\{ \Diamond_{m+(p+q)-t} C|\Diamond_m C \in P(t) \text{ and } p+q \geq t \} \cup \\
\{ \Diamond_{m-(t-(p+q))} C|\Diamond_m C \in P(t) \text{ and } t > p+q \} \cup \\
\{ \Diamond_{(t+m)-(p+q)} C|\Diamond_m C \in P(t) \text{ and } t+m \geq p+q \}
\]

(d) idem to item 9c replacing \( \Box(B \rightarrow \Diamond_p H) \in P(t) \) by \( \Box(B \rightarrow \Diamond_p H) \in P(t) \) and for the construction of \( P(s_2) \) and \( G(s_2) \) we must consider that:

- If \( q - p \geq 0 \) then we replace \( p + q \) by \( q - p \).
- If \( p - q > 0 \) then we replace \( p + q \) by \( p - q \), we apply the mirror image rule to the set \( P(s_2) \) and \( G(s_2) = \Diamond_{n-(p-q)} A \).

(e) idem to item 9d replacing \( \Box(B \rightarrow \Diamond_p H) \in P(t) \) by \( \Box(B \rightarrow \Diamond_p H) \in P(t) \), for the construction of \( P(s_2) \) and \( G(s_2) \) we must consider that:

- If \( p - q \geq 0 \) then we replace \( p + q \) by \( q - p \).
- If \( q - p > 0 \) then we replace \( p + q \) by \( p - q \), we apply the mirror image rule to the set \( P(s_2) \) and \( G(s_2) = \Diamond_{n+(q-p)} A \).

(f) idem to item 9c replacing \( \Box(B \rightarrow \Diamond_p H) \in P(t) \) by \( \Box(B \rightarrow \Diamond H) \in P(t) \), \( P(s_2) \) is the same applying first the mirror image rule and \( G(s_2) = \Diamond_{p+q+n} A \).

(g) idem to item 9c replacing \( \Box(B \rightarrow \Diamond_p H) \in P(t) \) by \( \Box(B \rightarrow H) \in P(t) \) and for the construction of \( P(s_2) \) we consider \( p = 0 \).

(h) idem to item 9g replacing \( \Box(B \rightarrow H) \in P(t) \) by \( \Box(B \rightarrow H) \in P(t) \) applying the mirror image rule to \( P(s_2), G(s_2) = \Diamond_{q+n} A \).

10. If \( t \) is not an endpoint, \( X(t) = 0 \) and \( G(t) = \Diamond_{n} A \), then \( t \) has exactly one immediate successor: \( s \) and the next condition holds: \( P(s) \) is the same as the one in 9, \( X(s) = 0 \) and \( G(s) = A \).

11. If \( t \) is not an endpoint, \( X(t) = 1 \) and \( G(t) = \Diamond_{n} A \), then \( t \) has exactly one immediate successor \( s \) and the condition of the item 9 applying the mirror image rule.

12. If \( t \) is not an endpoint, \( X(t) = 0 \) and \( G(t) = \Diamond_{n} A \), then \( t \) has exactly one immediate successor \( s \) and the condition of the item 10 applying the mirror image rule.

13. If \( t \) is not an endpoint, \( X(t) = 1 \) and \( G(t) = \Diamond A \), then \( t \) has only one successor, \( s_1 \) and the following condition holds: \( P(s) = P(t), X(s) = 1 \) \( \text{and} \ G(s) = \Diamond_{n} A \)

or it has two immediate successors, \( s_1 \) and \( s_2 \), and one of the following conditions holds:

(a) \( P(s_1) = P(t) \) and \( X(s_1) = 1 \) and for some \( H: G(s_1) \rightarrow \Diamond H \in P(t), X(s_2) = 1, \ G(s_2) = A \lor \Diamond A \text{ and} \)

\[
P(s_2) = \{ \Box C|C \in P(t) \} \cup \{ H \} \cup \{ \Box C|C \text{ is an ordinary clause in } P(t) \}
\]
(b) idem to item 13a replacing $G(s_1) \rightarrow \Diamond H \in P(t)$ by $\Box(G(s_1) \rightarrow \Diamond H) \in P(t)$
(c) idem to item 13b replacing $\Box(G(s_1) \rightarrow \Diamond H) \in P(t)$ by $\Box(G(s_1) \rightarrow H) \in P(t)$

14. if $t$ is not an endpoint, $X(t) = 0$ and $G(t) = \Diamond A$, then $t$ has $k$ immediate successors, $s_1, \ldots, s_k$ and for $m, n$ when ever $1 \leq m \leq n \leq k$ and the set of wff (well formed formulas) $D_1, \ldots, D_k$, $D_i(i \leq m)$ where these are all the clauses of the form $\Box(B_i \rightarrow H_i)$ in $P(t)$, $D(m+1), \ldots, D(n)$ are exactly all the clauses of the form $\Box(B_i \rightarrow \Diamond H_i)$ in $P(t)$, $D(n+1), \ldots, D(k)$ are exactly all the clauses of the form $B_i \rightarrow \Diamond H_i$ in $P(t)$, and for each $1 \leq i \leq k$ one of the following conditions holds:

(a) $P(s_i) = P(t)$, $X(s_i) = 0$ and $G(s_i) = B_i$

(b) $P(s_i) = \{\Box C | C \in P(t)\} \cup \{H_i\} \cup \{\Box C | C \text{ is an OC in } P(t)\} \cup \{C| \Diamond_n C \text{ is an OC in } P(t)\}$,

$X(s_i) = 0$,
$G(s_i) = A \lor \Diamond A$, for $1 \leq i \leq k$, except when $1 \leq i \leq m$, in which case $G(s_i) = \Diamond A$

15. if $t$ is not an endpoint, $X(t) = 1$ and $G(t) = \Diamond A$, then $t$ has one immediate successor, $s_1$ or has exactly two: $s_1, s_2$ and one of the conditions of item 13 holds, applying first, to them, the mirror image rule.

16. if $t$ is not an end point, $X(t) = 0$ and $G(t) = \Diamond A$, then similar conditions to the ones in item 14 (the same applying the mirror image rule) must hold.

17. If $t$ is not an endpoint, $X(t) = 1$ and $G(t) = A_1 \lor A_2$, then $t$ has exactly one immediate successor : $s$ with $P(s) = P(t)$, $X(s) = 1$ and $G(s)$ is either $A_1$ or $A_2$.

18. If $t$ is not an endpoint, $X(t) = 0$ and $G(t) = A_1 \lor A_2$, then $t$ has exactly two immediate successors : $s_1$ and $s_2$ with $P(s_i) = P(t)$, $X(s_i) = 0$ and $G(s_i) = A_i$.

$\square$

It is important to see that queries like $\Diamond p$ and $\Box p$, potentially demanding the exploration of the temporal structure, are handled in a finite sequence of steps. This is achieved because on one side each program has a finite number of facts and rules:

**Lemma 1** Let $P(n)$, $n \geq 0$, be a program in $MTPL$ at $n^{th}$ step of computation, $P(n)$ has always a finite set of clauses. $\blacksquare$

**Proof 2** Let $n_1$ be the number of facts, $n_2$ the number of Ordinary Rules, $n_3$ the number of Always Rules, $k_1$ the number of inferred facts and $k_2$ the number of inferred rules in each step.

If $P(0)$ is the initial program in $MTPL$ then $|P(0)| = n_1 + n_2 + n_3$. To calculate the cardinality of $P(t)$ we must be aware of an additional fact, according to the syntax of the language we only allow the use of $\Box$ and $\Box$ in the outermost part of a rule. Since Always Rules cannot be a consequent of any rule they cannot be obtained by using MP, and so cannot be deduced by Modus Ponens. The consequences of the implications can only be facts or Ordinary Rules. It is important to note that the algorithm uses only those heads of rules in the program to look for a possible way to answer the query. This is done much in the way Prolog does, so a finite and restricted amount of facts are considered in each step $n$ and $k_1$ and $k_2$ must have an upper bound given by the size of the program. We have then $|P(t)| = (n_1 + k_1) + (n_2 + k_2) + n_3$, $k_1, k_2 \geq 0$. $\square$
and the second reason is that information is handled using the following conventions: $\Diamond p$ is true at $t$ either if $p \in \sigma(t+n)$, $n \geq 0$, or if $\Diamond p \in \sigma(t)$ while $\Box p$ is true at $t$ $\Box p \in \sigma(t)$. This results in the labelled computation tree having a finite number of nodes in each step:

Theorem 4 Each labelled computation tree for MTPL has a finite set of nodes.

Proof We already know that each step is based on the facts and rules that are in $P(t)$:
1) $G(t)$ is an atom, we need to explore all the facts and rules of $P(t)$ in the worst case. This means that we could need $|P(t)|$.
2) $G(t)$ is a formula like $A \wedge B$ in the worst case, $x(t) = 1$, we will have 2 son nodes.
3) $G(t)$ is a formula of the form $A \vee B$ in the worst case, $x(t) = 0$, we will also have 2 son nodes.
4) $G(t)$ has the form $\neg A$ in the worst case it will have only one successor node in the tree.
5) $G(t)$ has the form $\Diamond_{n} A$ (analogously for $\Box_{n} A$), in the worst case it cannot be deduced in the instant referred by the operator and the tree will explore each different way to deduce as the consequent of an ordinary or an always rule. In the worst case each rule of $P(t)$ could have a direct or indirect reference to $\Diamond_{n} A$ so the amount of nodes to consider is $|P(t)|$.
6) $G(t)$ has the form $\Box A$ (analogously for $\Box A$), which leads to the same situation as in the previous item.

For each query the tree will have a finite set of nodes, and each node has a finite set of facts and rules (see lemma 1). □

We must point out however the following hypothesis: a) This result concerns the algorithm, not a particular implementation. For example, if we use Prolog, it will be enough to consider a function or omit occurs check to obtain the possibility of infinite computations. b) we assume the program has no cycles introduced by the programmer, to say, $A \rightarrow ..., ..., \rightarrow A$

It is also interesting to see that all the operators seen in section 2.2 lead to a finite sequence of computations because they can be rewritten using the definitions provided previously in terms of the operators considered in the basic language $\mathcal{M}$.

4 Some examples

We show in this section how some situations demanding a combination of order and duration concepts can be handled through MTPL. This combination appears naturally in several contexts but we give some examples to show its applicability to Real time system, multimedia synchronization and temporal databases constraints specification.

Example 1 “If in some moment in the future the temperature of the tank reaches a critical temperature then in 10 minutes the valve must open”. Take as a hypothesis a temporal granularity of minutes, the above affirmation will be represented in MTPL as:

$$\Box (\text{critic\_temperature} \rightarrow \Diamond_{10} \text{open\_valve})$$

Example 2 “The sound of the video does not begin after 5 seconds of the beginning of the emission of the image”. Assuming a temporal granularity of seconds, this could be expressed as:

$$\text{begin\_to\_emit\_image} \rightarrow \Box_{[0,5]} \neg \text{emit\_sound}$$

using the operators, presented in section two, for handling periods.
Example 3 Another thing that we want to represent is that a person is considered as an employee, since he is contracted until he is fired or he resigns. So we have:

$$\Diamond_{[a,b]} \text{employee}(X) \rightarrow \Diamond_a \text{contract}(X) \land \Diamond_b (\text{fired}(X) \lor \text{resigns}(X))$$

The examples we already saw illustrate some way formulas that can be expressed in our language. Now we are going to see an example of the system’s behaviour.

Example 4 Let us consider the following program:

Rules: \[\begin{align*}
(1) & \quad \Box (\text{working}(X) \land \neg \text{critic.temperature}(X) \rightarrow \Diamond_1 \text{working}(X)) \\
(2) & \quad \Box (\text{working}(X) \land \text{critic.temperature}(X) \rightarrow \Diamond_1 \text{out.of.service}(X) \land \\
& \quad \Diamond_2 \text{out.of.service}(X) \land \Diamond_3 \text{out.of.service}(X) \land \\
& \quad \Diamond_4 \text{out.of.service}(X) \land \Diamond_5 \text{working}(X))
\end{align*}\]

Facts: \[\begin{align*}
(3) & \quad \text{working(Machine1)} \\
(4) & \quad \text{working(Machine2)} \\
(5) & \quad \Diamond_3 \text{critic.temperature}(\text{Machine1}) \\
(6) & \quad \Diamond_4 \text{out.of.service}(\text{Machine1})
\end{align*}\]

We have the following queries

Query1: $\Diamond_{15} \text{working}(\text{Machine3})$
Query2: $\Diamond_1 \text{out.of.service}(\text{Machine1})$

The answer to Query1 must be no, and the one for Query2 must be yes.

Let consider the first one. The set $P(0)$ is the programme as it appears above. $G(0)$ has the form $\Diamond_6 A$, so the algorithm chooses rule 9. From the alternatives that has in that item, it chooses the first possibility (one successor) then $P(15)$ is contracted in the way specified by the algorithm, $G(15) = \text{working}(\text{Machine3})$ which is an atom. But we can see that $G(15) \not\in P(15)$ so it fails. As it fails the algorithm will try to solve the query choosing one of the two successors of the item but none of them is applicable. Since none of the clauses can satisfy the query, they cannot deduce $\Diamond_1 \text{working}(\text{Machine3})$. The algorithm has no other possible choice so the answer to the query is no, as expected.

On the other hand if we consider Query2 the algorithm again can answer it affirmatively choosing rule 9 because of fact (6). But if we consider the programme without that fact the algorithm applies anyway rule 9, but this time using one of the two successors. Because of the set of rules we are considering in this example, it must choose item (g) over program’s rule (2). According to that item

$$G(s_1 = \Diamond_q (\text{working}(\text{Machine1}) \land \text{critic.temperature}(\text{Machine1}))$$

as we are going to see $q = 3$ and so $G(s_2 = \Diamond_1 \text{out.of.service}(\text{Machine1})$ which is trivially satisfied because this fact belongs to $(P)(s_2)$.

Now let us see how the algorithm finds that $q = 3$. Remember that the query is $G(s_1 = \Diamond_q (\text{working}(\text{Machine1}) \land \text{critic.temperature}(\text{Machine1}))$ for some $q$ the algorithm through item 9 try to satisfy $G(s_1)$ for $q = 1$ and $q = 2$ but fails because it cannot deduce $\text{critic.temperature}(\text{Machine1})$ in none of those points. It tries to resolve it for $q = 3$. According to item 9 and the one-successors choice it builds the base for

$$G(s_1') = \text{working}(\text{Machine1}) \land \text{critic.temperature}(\text{Machine1}).$$

This new query is solved through item 6 in the algorithm, which gives us two new queries that are trivially satisfied because they are atoms in the corresponding set $P$. So finally the algorithm answers yes. \qed
5 Conclusions and Future Work

We have provided a temporal programming language, MTPL, based on a metric temporal logic. We have shown that MTPL is a proper extension of “Temporal Prolog” [Gab87] mixing at the same time the simplicity and naturality of temporal operators with the possibility to refer to time in a more direct way. Some examples of these benefits can be seen in [CA99b] where we used MTPL to handle appropriately interesting situations in temporal databases that other proposals cannot. In [CA99c] we show that we can also use MTPL to express synchronization conditions in multimedia in a convenient way. An algorithm based on the notion of labelled computation tree inspired by [Gab87] was offered which could be used for the future implementation of an interpreter for queries to MTPL. Our current research agenda includes the study of other metatheoretical properties and further comparisons with different languages inspired in similar aims.

References


