Real-Time Systems, 2, 255–299 (1990)

Specifying Real-Time Properties with Metric Temporal Logic

RON KOYMANS*
Philips Research Laboratories, P.O. Box 80.000, 5600 JA Eindhoven, The Netherlands

Abstract. This paper is motivated by the need for a formal specification method for real-time systems. In these systems *quantitative* temporal properties play a dominant role. We first characterize real-time systems by giving a classification of such quantitative temporal properties. Next, we extend the usual models for temporal logic by including a distance function to measure time and analyze what restrictions should be imposed on such a function. Then we introduce appropriate temporal operators to reason about such models by turning qualitative temporal operators into (quantitative) metric temporal operators and show how the usual quantitative temporal properties of real-time systems can be expressed in this metric temporal logic. After we illustrate the application of metric temporal logic to real-time systems by several examples, we end this paper with some conclusions.

1. Introduction

This paper is motivated by the need for a formal specification method for real-time systems. The need for such a method is becoming acute since more and more vital applications such as nuclear power stations, computer controlled chemical plants, flight control software for airplanes, etc., contain real-time features. Real-time systems are characterized by *quantitative* timing properties relating occurrences of events. Typical examples are:

1. Maximal distance between an event and its reaction, for example, every A is followed by a B within 3 time units (a typical promptness requirement).
2. Exact distance between events, for example, every A is followed by a B in exactly 7 time units (as with the setting of a timer and its time-out).
3. Minimal distance between events, for example, two consecutive As are at least 5 time units apart (assumption about the rate of input from the environment).
4. Periodicity, for example, event E occurs regularly with a period of 4 time units.
5. Bounded response time, for example, there is a maximal number of time units so that each occurrence of an event E is responded to within this bound.

We investigate the possibilities of temporal logic for specifying such real-time properties. Because they only involve qualitative temporal operators, it is obvious that standard temporal logics cannot deal with quantitative temporal requirements. Therefore, we extend the usual temporal models by including a distance function to measure time and analyze what restrictions should be imposed on such a function. This distance function maps two points in

*Part of this research has been performed at the Eindhoven University of Technology when the author was working in ESPRIT project 937: Debugging and Specification of Ada Real-Time Embedded Systems (DESCARTES).

time to a value in a metric domain on which addition and a zero are defined. The specification method we propose, called *metric temporal logic*, is based on an extension of the classical temporal logic as studied in philosophy for decades (see for example, (Prior 1967)): our metric operators are obtained by indexing temporal operators by parameters taken from the metric domain. Our philosophy is to extend the pure qualitative view of time of standard temporal logics in a faithful way in order to reason both about quantitative and qualitative properties in a convenient way. We succeed in doing this by including also the precedence relation between points in time and showing how the metric parameters of operators can be quantified away to obtain the corresponding qualitative versions. We show how the five quantitative timing properties above can be expressed in metric temporal logic.

We illustrate metric temporal logic by means of several examples involving real-time features amongst which are common real-time constructs such as a time-out and the wait/delay statement of some concurrent programming languages.

This paper is organized as follows. In Section 2 we describe certain characteristics of real-time systems. After a short recapitulation of the basics of temporal logic in Section 3, Section 4 introduces metric temporal logic which is illustrated by means of a series of specification examples of real-time systems in Section 5. At last we present some conclusions in Section 6.

2. What are real-time systems?

The most important characteristic of a real-time system is the demand to keep abreast with an autonomous environment by reacting properly and timely to events occurring in the environment asynchronously from the operation of the system. Therefore, the environment-system interaction (the reaction of the system on the external stimuli from the environment giving rise to a so-called stimulus-response mechanism) is subject to quantitative temporal requirements. These temporal requirements state a relation between occurrences of events and can be classified as follows:

1. Response time: this relates the timing of the occurrence of an event and the resulting response. The most usual cases are
   a. maximal distance between an event and the resulting response (e.g., time-out)
   b. exact distance (e.g., delay)
2. Frequency: this relates occurrences of the same event. The most usual cases are
   a. minimal distance between two occurrences (assumption about the rate of stimuli from the environment)
   b. exact distance, also called periodicity (e.g., clocks and samplers)

The first four of the five examples in the Introduction correspond directly to the classification above (examples 1 and 2 concern maximal respectively exact response time and examples 3 and 4 concern minimal respectively exact frequency). All these temporal requirements have a quantitative nature and the quantitative elements involved are constants expressed as a certain number of time units. The fifth example in the Introduction is in fact the quantified equivalence of the first example. The other examples 2, 3 and 4 have also quantified

equivalents, but example 5 is the most common one. The quantitative nature of these temporal requirements is typical for real-time systems (qualitative temporal requirements occur already in any concurrent system, think of fairness, and even sequential systems, e.g., termination).

Another classification of quantitative temporal requirements relates to the distinction between relative and absolute temporal requirements. Absolute temporal requirements calibrate all occurrences of events to a fixed reference point (the start of the system or the first occurrence of a particular event) while relative temporal properties have no fixed reference point but depend on occurrences of events. In the above four cases, periodicity is an absolute temporal requirement (e.g., all later samples can be related to the first sample by means of the sample rate), the other three being relative (the occurrence of an event triggers its response, so the timing of that response can only be related to that occurrence of the event). As will be clear from the above, events play a very important role in real-time systems.

Since quantitative temporal requirements state a relation between an event in the environment and an event in the system (or between events in different components of a system), these requirements necessarily refer to a global notion of time. This global notion of time should not be identified with the introduction of a clock: the difference between time and clocks is that clocks always drift (in other words: time can be considered as a perfect, idealized clock).

In the theory of concurrency the interleaving model plays an important role. One can question the suitability of this model in case real-time and concurrency are combined. Modeling parallel computation by interleaving is a sufficient idealization if only qualitative temporal requirements are involved. As soon as quantitative temporal requirements come into play, however, as in the case of real-time systems, such an execution model is usually not adequate any more. For example, ensuring maximal distance between events is impossible if some processes can take an arbitrary number of steps while other processes are inactive. In such a case either all processes have their own processor (the maximal parallelism model as in (Koymans, Shyamasundar, de Roever, Gerth and Arun-Kumar 1985)) or some processes share one processor and they are scheduled in such a way that each process gets its turn within bounded time. Furthermore, in some applications, data can appear at different places in a truly concurrent way. With respect to the temporal requirements above, an arbitrary sequentialization is not appropriate any more. Even stronger, it becomes more and more practice today to incorporate local (co)processors with dedicated tasks (e.g., sampling) into the system so that truly parallel computation is the only realistic model in such a distributed configuration.

In process control systems often continuous physical entities are involved such as temperature and volume. When such a system contains, for example, an analog circuit for monitoring the temperature, this has a time-continuous nature together with a continuous range of values (e.g., between 4 and 20 milliampere). In modeling such systems, the usual discrete view of time as taken for digital systems is, therefore, not appropriate any more. Hence, apart from viewing time as discrete, one should also allow a view of time as continuous (or at least dense) as in Newtonian physics. This has also its repercussion on the description of the execution of such a system (or rather how it develops) and how it can be observed. For discrete systems, execution consists of a number of observable state changes or transitions leading to a state-transition sequence. In the case of time-continuous systems, however,

variables can change arbitrarily fast (think, e.g., of pressure) and sequences cannot be used any more. A particular execution can only be described by recording at *each* moment the state of the system (so, such a generalized execution model considers functions from time to states). If one would maintain that observations can be made only at discrete moments, each observation contains only partial information. Only the whole set of possible observations of a particular execution can restore all information on that execution.

Summarizing, for real-time systems quantitative temporal requirements play a dominant role. Furthermore, a discrete view of time and familiar execution models such as interleaving are not sufficient any more to handle all cases. Consequently, time-continuous models, respectively real parallelism or scheduling information should be incorporated.

We devote the rest of this section to the subject of requirements for a specification language for real-time properties. First, we want our specifications to be free of any implementation bias whatsoever. This means that a specification is phrased only in terms of the elements of the interface (between the specified entity and its environment) that are considered to be observable. This leads to the requirement of syntactical abstractness (see (Koymans 1989)). In case of the specification of real-time properties syntactical abstractness requires that the specification of temporal properties is stated only in terms of the events involved and the relevant quantity of time units.

As a second requirement, we want the specification language to be formal in order to ensure rigorous analysis and verification of desired properties. Further advantages of a formal approach include:

- in the process of formalization ambiguities, omissions and contradictions in the informal requirements can be detected,
- a formally verified part can be embedded with more confidence that it will function correctly (the formal model leads to enhanced reliability),
- the formal model can be a foundation for (partly) automated design methods and tools such as simulators,
- several designs can be compared.

The introduction of formal methods for real-time systems has lagged behind that for other application areas. Most specification methods do not include constructs to express timing in a quantitative way and the few syntactical formalisms that include timing, lack formal semantics. It is thus not surprising that only a minority of suitable formal methods for real-time systems have been developed. In fact, most of them were developed during the last few years. Some of these methods do not tackle all problems of real-time systems but concentrate for example, on discrete event systems. Several reasons can be given for the fact that formal methods for real-time systems lag behind that for other application areas:

1. Because the timing requirements are much stricter for real-time systems than for other systems, they impose more demands on the implementation technology; therefore, implementation concerns (e.g., processor speed) were dominant in the era before the explosive growth of computing power for microprocessors that started about ten years ago,
2. The intrinsic complexity of typical real-time systems makes it much more difficult to develop adequate formal methods,

3. Most researchers in theoretical computer science have considered real-time either as a special (though admittedly harder) case of concurrent systems, or as a topic whose study should be postponed until we understand basic concurrency better.

3. Classical temporal logic

In this section we first recapitulate the basics of the propositional temporal logic used in philosophy since decades (see e.g., (Prior 1967)). We start out from a propositional language containing proposition letters ($p, p_1, p_2, \ldots, q, \ldots$), two propositional constants $\bot$ (falsum) and $\top$ (verum), and the boolean operators $\neg$ (not), $\wedge$ (and), $\vee$ (or), $\rightarrow$ (if ... then ...) and $\leftrightarrow$ (if and only if). From these proposition letters, propositional constants and boolean operators, formulas $\varphi, \varphi_1, \ldots, \psi, \psi_1, \ldots$ are built in the usual way.

To this propositional language temporal operators can be added. The temporal logic (in philosophy also known as tense logic) of (Prior 1967) adds four operators: **G** (it is always going to be the case), **F** (at least once in the future), **H** (it has always been the case) and **P** (at least once in the past). For a unary operator **O** its dual $\overline{\mathbf{O}}$ is defined by

$$\overline{\mathbf{O}}\,\varphi := \neg\,\mathbf{O}\,\neg\,\varphi.$$

(Then $\overline{\mathbf{O}_1\,\mathbf{O}_2}$ equals $\overline{\mathbf{O}_1}\,\overline{\mathbf{O}_2}$ and $\overline{\overline{\mathbf{O}}}$ equals **O**.) The pairs **G**, **F** and **H**, **P** are duals of each others.

The semantics of temporal logic is based on frames and models:

DEFINITION 3.1. A frame (also called point structure) is a pair $(T, <)$ where T is a nonempty set of 'moments' (points in time) and $<$ is a binary relation on T ('precedence' or 'earlier' relation).

A model is a triple $(T, <, V)$ where $(T, <)$ is a frame and V is a valuation on T, i.e., it maps proposition letters onto subsets of T (giving the set of moments where this proposition holds).

DEFINITION 3.2. A temporal formula φ holds in $\mathfrak{M} = (T, <, V)$ at $t \in T$, notation $\mathfrak{M}, t \vDash \varphi$, is defined by recursion:

$$\begin{aligned}
&\mathfrak{M}, t \vDash p && \text{iff } t \in V(p) \text{ (for any proposition letter } p)\\
&\mathfrak{M}, t \vDash \bot && \text{for } \textit{no } \mathfrak{M} \text{ and } t\\
&\mathfrak{M}, t \vDash \varphi \rightarrow \psi && \text{iff } \mathfrak{M}, t \vDash \varphi \Rightarrow \mathfrak{M}, t \vDash \psi\\
&\mathfrak{M}, t \vDash \mathbf{G}\varphi && \text{iff } \forall t' \in T\,[t < t' \Rightarrow \mathfrak{M}, t' \vDash \varphi]\\
&\mathfrak{M}, t \vDash \mathbf{H}\varphi && \text{iff } \forall t' \in T\,[t' < t \Rightarrow \mathfrak{M}, t' \vDash \varphi].
\end{aligned}$$

From this definition two further important notions of validity (validity in a model, respectively universal validity) can be derived as follows:

$\mathfrak{M} \models \varphi$ if $\forall t \in T\ \mathfrak{M}, t \models \varphi$

$\models \varphi$ if for all models $\mathfrak{M}$ $\mathfrak{M} \models \varphi$.

Next we study temporal logics with **until** and **since** operators. We first define the syntax of propositional temporal logic with **until** and **since** operators.

DEFINITION 3.3. L(**until**, **since**) is the language with

vocabulary: atomic propositions $P_0, P_1, \ldots$
logical operators $\neg$, $\wedge$, **until**, **since**

formulas: $P_i (i \in \mathbb{N})$
$\neg\ \varphi_1$, $\varphi_1 \wedge \varphi_2$, φ_1 **until** φ_2 and φ_1 **since** φ_2 (φ_1, φ_2 formulas).

To give the semantics of L(**until**, **since**) we can use the notions of frames, valuations and models introduced before (see Definition 3.1). For languages with **until** and **since** we will suppose the temporal frames to be strict partial orders, that is, $<$ is transitive and irreflexive. In the definition of $\mathfrak{M}, t \models \varphi$ we have only to include the following clauses for the operators **until** and **since**:

$$\mathfrak{M}, t \models \varphi_1 \textbf{ until } \varphi_2 \quad \text{iff} \quad \exists t' \in T[t < t' \text{ and } \mathfrak{M}, t' \models \varphi_2 \text{ and } \forall t'' \in T[t < t'' < t' \Rightarrow \mathfrak{M}, t'' \models \varphi_1]]$$

$$\mathfrak{M}, t \models \varphi_1 \textbf{ since } \varphi_2 \quad \text{iff} \quad \exists t' \in T[t' < t \text{ and } \mathfrak{M}, t' \models \varphi_2 \text{ and } \forall t'' \in T[t' < t'' < t \Rightarrow \mathfrak{M}, t'' \models \varphi_1]].$$

Because of irreflexivity of $<$ the operators **until** and **since** will also be irreflexive, that is, they do not include the present as part of the future.

The temporal operators **F**, **G**, **P**, **H** can be defined easily in terms of **until** and **since**:

$\mathbf{F}\varphi := \top$ **until** φ

$\mathbf{P}\varphi := \top$ **since** φ

where still $\mathbf{G} \equiv \overline{\mathbf{F}}$ and $\mathbf{H} \equiv \overline{\mathbf{P}}$, of course.

Similarly to the extension of propositional logic with the temporal operators **until** and **since** one can extend predicate logic with these operators to get a first-order temporal logic. In first-order temporal logics, problems arise because of the interplay between quantification and time (see e.g. (Garson 1984; Cocchiarella 1984)). One of these problems is the possibility that the quantified variables (and possibly even their value domains) change over time. We avoid this problem by only allowing quantification over variables that do not change over time (often called global variables in contrast with local variables). Even in this restricted case most first-order temporal logics are incomplete (usually shown by proving that Peano Arithmetic can be encoded into them).

We end this recapitulation of temporal logic by a brief sketch of some issues involving the application of temporal logic in computer science. Since the seminal paper of Pnueli

(1977) the use of temporal logic for reasoning about many types of computerized systems and programs has been steadily increasing. This can be explained by the fact that the underlying semantics of temporal logic fits well with the notion of computation as used in computer science as we will show now. Temporal logic is intended for reasoning about situations changing in time. Its semantics makes a clear distinction between the static aspect of a situation, represented by a state, and the dynamic aspect, the relation (in time) between states. This distinction is also reflected in the syntax: a state is described by the classical part of temporal logic while the temporal operators are used for the description of the evolution of the situation over time. In this way states and time need not be introduced explicitly in the logic itself. The connection with the notion of computation is that a computation can be seen as a sequence of states where each transition from one state to the next state in the sequence (each step of the computation) can be thought of as a tick of some computation clock. In this view computer systems are described as generators of computations (also called execution sequences). Therefore, the applications of temporal logic in computer science are usually restricted to the class of discrete systems where an execution of a system can be viewed indeed as a sequence of state transitions. For that reason the temporal frames considered are also discrete.

The two most common types of temporal frames used in computer science are the natural numbers with their usual ordering and tree-like structures where branching is allowed only towards the future, giving rise, respectively, to what is commonly called linear (time) temporal logic and branching time temporal logic. For a comparison between linear and branching time temporal logic, see for example, (Stirling 1987). Apart from linear and branching time temporal logic, there are temporal logics in use in computer science that are based on other types of temporal frames, for example, the partial order temporal logic of (Pinter and Wolper 1984), the temporal logic for event structures of (Penczek 1988) and the interleaving set temporal logic (using a mixture of branching time and partial order elements) of (Katz and Peled 1987), but these form only a minority. Another approach is where temporal frames are not based on points but on intervals instead. This approach is also represented in computer science (for an excellent overview of the interval-based approach versus the point-based approach in philosophy see (Benthem 1983)), for example, Interval Temporal Logic with its executable subset Tempura of Moszkowski (see (Moszkowski 1983; Moszkowski and Manna 1984; Moszkowski 1986)) and the interval logic of Schwartz et al. (Schwartz, Melliar-Smith and Vogt 1983).

In this paper we restrict our attention to temporal logics based on temporal frames with a precedence relation that is linear, in other words we look only at linear time-like temporal logics. (Pnueli 1977) contains some deviations from classical temporal logic (as treated before), in particular:

1. the present is considered as part of the future and correspondingly the basic temporal operators are reflexive,
2. only future temporal operators are used.

In the sequel we will denote the reflexive counterparts of the temporal operators **F** and **G** by their usual representation in computer science $\Diamond$, respectively $\Box$. In general, irreflexive temporal operators have more expressive power than the corresponding reflexive ones.

Although not done in (Pnueli 1977) several later papers have included the operators **X** (next) and **Y** (previous) for indicating the next, respectively previous, element in the precedence relation (remember that now this relation is supposed to be a discrete ordering). Over the natural numbers the irreflexive operators can then be expressed, for example, $\mathbf{F}\varphi \equiv \mathbf{X}\Diamond\varphi$. The operators **X** and **Y** also have their deficiencies, however. For example, these operators lack the abstractness needed to achieve a fully abstract semantics (i.e., on a level of abstraction equalling that of a semantics formalizing the chosen notion of observable behavior) of concurrent programs (see (Lamport 1983a; Barringer, Kuiper and Pnueli 1986)).

Concerning the second deviation above, it can be shown that the temporal operator **until** already suffices for expressive completeness over the natural numbers. Therefore, from the viewpoint of expressive power there is no need to introduce past operators when working over the natural numbers. However, (Koymans, Vytopil and de Roever 1983) showed the advantages of such operators for the elegant specification of message passing systems and (Lichtenstein, Pnueli and Zuck 1985) contains many theoretical results about the usefulness of past operators.

We now come back on the topic of temporal logic as a specification language for computerized systems and programs. As we have seen above, a computation of a computer system can be described as a (linear) sequence of states and associated events (state transitions). In linear temporal logic the approach is taken that the behavior of a system S is given by the set of its computations, say Σ. A temporal formula φ is then defined to be valid for S (φ is a valid property of S) if each $\sigma \in \Sigma$ satisfies φ, that is, $\sigma \models \varphi$ in the sense of Definition 3.2 above (remember that the underlying time domain of linear temporal logic is the set of natural numbers so that the sequence σ can function as a model).

Data elements (e.g., those exchanged between a system and its environment) can be partitioned into state variables and events. For the description of data elements we need a first-order variant of linear temporal logic. This variant partitions the set of variables into so-called global and local variables where quantification is only allowed over global variables (so the local variables always occur as free variables). Global variables range over fixed data domains and serve to denote elements thereof while local variables model the state variables (such as variables occurring in a program). Events are modeled as predicates (where the parameters of the event become the arguments of the predicate).

When using temporal logic for the specification of programs, a fundamental classification of program properties differentiates between safety- and liveness-properties. For a syntactical classification of temporal properties into a hierarchy refining this safety-liveness classification, see (Manna and Pnueli 1987). Characterizations and decidability of safety- and liveness-properties using connections with model theory, formal language theory and semigroup theory are contained in (Thomas 1986).

To end our account of the application of temporal logic as a specification language in computer science, we can test temporal logic against the requirements for a specification language for real-time properties in Section 2. Syntactical abstractness can be achieved by restricting the local variables and predicates to the externally observable state variables and events, respectively. This section witnesses the formality of temporal logic. In fact, temporal logic is a simple and elegant extension of propositional logic (predicate logic in case of first-order temporal logic), yet powerful enough to express interesting properties of programs such as safety- and liveness-properties. Furthermore, papers such as (Lamport

1983b; Barringer, Kuiper and Pnueli 1984; Barringer and Kuiper 1985a, b) show that temporal logic can be used for hierarchical development in a compositional and modular style.

4. Metric temporal logic

In this section we look at ways of reasoning with temporal logic about quantitative timing properties such as those mentioned in the Introduction. The standard models for temporal logic based on point structures involve a pure qualitative view of time by considering only a set of moments T together with the precedence relation $<$ (see Definition 3.1). The question now is: What should be added to such point structures $(T, <)$ to be able to handle also quantitative temporal properties? Because the evaluation of formulas is dependent on a particular point in time (representing the present), we suggest that apart from the precedence relation between the present and other points in time also the distance between points in time is needed. Therefore we add a distance function d with the idea that $d(t, t')$ gives a measure as to how far t and t' are apart in time. The next question is: What conditions should be put on $<$ and d? Because quantitative temporal properties relating different components of a system must necessarily refer to a global conception of time, we assume that the set of time points can be ordered in a global way. So, we suppose that the precedence relation $<$ is total (i.e., transitive, irreflexive and comparable). For the distance function d we suppose the usual topological conditions apart from the replacement of the triangular inequality by a conditional equality:

(d1) $d(t, t') = 0 \Leftrightarrow t = t'$

(d2) $d(t, t') = d(t', t)$

(d3) if $t < t' < t''$ then $d(t, t'') = d(t, t') + d(t', t'')$ and $d(t'', t) = d(t'', t') + d(t', t)$.

Next we should determine the range of d. There is no reason to choose the standard reals (in fact, the example below shows the usefulness of nonarchimedean ranges for d). As is apparent from the conditions (d1)–(d3) above we need a structure with addition and zero element. So, we suppose as range for d a structure $(\Delta, +, 0)$ where addition $+$ and constant 0 are restricted by:

(Δ1) $\delta + \delta' = \delta' + \delta$	(commutativity)
(Δ2) $(\delta + \delta') + \delta'' = \delta + (\delta' + \delta'')$	(associativity)
(Δ3) $\delta + 0 = \delta = 0 + \delta$	(unit 0)
(Δ4) $\delta + \delta' = \delta + \delta'' \Rightarrow \delta' = \delta''$ and $\delta + \delta'' = \delta' + \delta'' \Rightarrow \delta = \delta'$	($+$ injective in both arguments)
(Δ5) $\delta + \delta' = 0 \Rightarrow \delta = 0$ and $\delta' = 0$	(no negative elements)
(Δ6) $\exists\delta''\,[\delta = \delta' + \delta''$ or $\delta' = \delta + \delta'']$	(existence of absolute difference).

In these conditions the free variables should be universally quantified (we left this out for the sake of concise presentation). One can easily check the independence of these restrictions on $(\Delta, +, 0)$, that is, that none of these restrictions follows from the others, by means of appropriate examples in which five of these restrictions hold and the sixth fails. An example is $\Delta = \mathbb{N} \cup \{e\}$ where we take over the standard addition for natural numbers supplemented by the following rules for the extra element e (which resembles 1):

$$e + e = 2,\ e + 0 = 0 + e = e \text{ and } e + n = n + e = n + 1 \text{ for } n \in \mathbb{N} \setminus \{0\}.$$

This structure $(\Delta, +, 0)$ obeys all restrictions $(\Delta 1)$–$(\Delta 6)$ above except $(\Delta 4)$:

$$e + e = e + 1, \text{ but } e \neq 1.$$

In spite of their independence, these restrictions nevertheless contain some redundancy (e.g., the second equality in $\Delta 3$ is added although this already follows from $\Delta 1$) in order to state the intended restriction fully also in the case when some of the other restrictions have been dropped. These conditions are motivated as follows. $(\Delta 1)$ is enforced by (d2) and (d3). One also needs to order Δ to compare different distances (think, e.g., of the expression of maximal distance, see point 1 in the Introduction). To this end, first define

$$\delta \underline{\ll} \delta' := \exists \delta''\ [\delta' = \delta + \delta''].$$

Such a δ'' is unique because of $(\Delta 4)$. Furthermore, $\Delta 2$ (transitivity) and $\Delta 3$ (reflexivity) make $\underline{\ll}$ a preorder. The corresponding irreflexive relation defined by

$$\delta \ll \delta' := \exists \delta''\ [\delta'' \neq 0 \text{ and } \delta' = \delta + \delta'']$$

is in fact a total order (comparable by $\Delta 6$) with 0 as its least element (by $\Delta 5$).

This leads to the following notion.

DEFINITION 4.1. A *metric point structure* is a two-sorted structure $(T, \Delta, <, d, +, 0)$ with signature $< \subseteq T \times T$, $d : T \times T \to \Delta$, $+ : \Delta \times \Delta \to \Delta$, $0 \in \Delta$ such that

(i) $<$ is total
(ii) d is surjective and satisfies (d1)–(d3)
(iii) $(\Delta, +, 0)$ satisfies $(\Delta 1)$–$(\Delta 6)$.

Δ and d are called the *metric domain* and the *temporal distance function*, respectively.

In (ii) surjectivity of d is demanded to get a nice correspondence between T and Δ. All these conditions on $<$ and d were motivated either by practical reasons (having a certain application area in mind) or by our wish to obtain a nice mathematical theory. Nevertheless, in some cases these conditions could be relaxed, for example it may be beneficial to allow a cluster of points having distance 0 to each other (deleting the only if case of condition d1). For the time being, we consider the above conditions as the most natural ones.

Example 4.1. Consider the following metric point structure.

$$T := \mathbb{N} \times \mathbb{N}$$

$$\Delta := \{0\} \times \mathbb{N} \cup \mathbb{N}^+ \times \mathbb{Z}$$

where $\mathbb{N}$, $\mathbb{N}^+$ and $\mathbb{Z}$ represent the natural numbers, the positive natural numbers, respectively the integers.

Define furthermore

$$(n, n') < (m, m') := n < m \text{ or } (n = m \text{ and } n' < m')$$

$$d((n, n'), (m, m')) := \begin{cases} (0, |n' - m'|) & \text{if } n = m \\ (m - n, m' - n') & \text{if } n < m \\ (n - m, n' - m') & \text{if } n > m \end{cases}$$

$$(n, z) + (n', z') := (n + n', z + z')$$

$$0 := (0, 0)$$

Figure 1 represents T together with its ordering $<$ (to be read from left to right). The idea is that the first component of T represents a kind of macro-time while the second component represents micro-time. It is easy to check that this example satisfies all conditions for a metric point structure and that the given Δ is nonarchimedean.

Having determined what the new temporal models should be, we now must find appropriate temporal operators for reasoning about metric point structures. This is done by transforming the temporal operators **G**, **F**, **H**, **P** from Section 3 into metric operators as follows (here and in the sequel $\mathfrak{M}$ denotes a metric model, that is, a metric point structure together with a valuation analogous to the notion of a model for classical temporal logic as defined in Definition 3.1):

$$\mathfrak{M}, t \vDash \mathbf{G}_\delta \varphi := \forall t' \in T[(t < t' \text{ and } d(t, t') = \delta) \Rightarrow \mathfrak{M}, t' \vDash \varphi]$$

$$\mathfrak{M}, t \vDash \mathbf{F}_\delta \varphi := \exists t' \in T[t < t' \text{ and } d(t, t') = \delta \text{ and } \mathfrak{M}, t' \vDash \varphi]$$

$$\mathfrak{M}, t \vDash \mathbf{H}_\delta \varphi := \forall t' \in T[(t' < t \text{ and } d(t, t') = \delta) \Rightarrow \mathfrak{M}, t' \vDash \varphi]$$

$$\mathfrak{M}, t \vDash \mathbf{P}_\delta \varphi := \exists t' \in T[t' < t \text{ and } d(t, t') = \delta \text{ and } \mathfrak{M}, t' \vDash \varphi].$$

Again $\mathbf{F}_\delta \equiv \overline{\mathbf{G}_\delta}$ and $\mathbf{P}_\delta \equiv \overline{\mathbf{H}_\delta}$. Formally, we use the standard first-order language (including identity $=$) over $(\Delta, +, 0)$ whose terms t are used to form the metric operators $\mathbf{G}_t$, $\mathbf{F}_t$, $\mathbf{H}_t$ and $\mathbf{P}_t$.

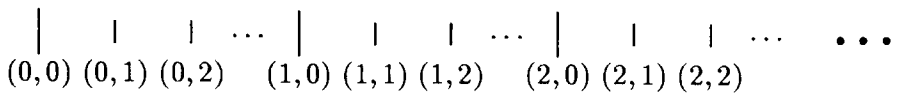


Figure 1. Example of a metric point structure: micro-macro time.

The metric operators $\mathbf{F}_\delta$ and $\mathbf{G}_\delta$ (and similarly for the pair $\mathbf{P}_\delta$ and $\mathbf{H}_\delta$) are strongly related. If we assume (d1)–(d3) and comparability of $<$ (both are true for metric point structures) it is easy to see that $\mathbf{F}_\delta$ can indicate at most one point (i.e., $\forall t \neg \exists t' t'' [t < t'$ and $t < t''$ and $t' \neq t''$ and $d(t, t') = d(t, t'') = \delta]$). Because $\mathbf{G}_\delta$ is the dual of $\mathbf{F}_\delta$ it must indicate the same point (if it exists). In fact, the existence of this point is exactly the difference between $\mathbf{F}_\delta$ and $\mathbf{G}_\delta$ ($\mathbf{F}_\delta$ asserts its existence while $\mathbf{G}_\delta$ does not) as is expressed by the syntactical equivalence

$$\mathbf{F}_\delta \varphi \equiv \mathbf{F}_\delta \top \wedge \mathbf{G}_\delta \varphi.$$

From the operators $\mathbf{F}_\delta$ and $\mathbf{P}_\delta$ several more metric operators can be derived:

$$\mathbf{F}_{<\delta} \varphi := \exists \delta' [0 \ll \delta' \ll \delta \wedge \mathbf{F}_{\delta'} \varphi]$$

$$\mathbf{P}_{<\delta} \varphi := \exists \delta' [0 \ll \delta' \ll \delta \wedge \mathbf{P}_{\delta'} \varphi]$$

$$\varphi \ \mathbf{until}_\delta \ \varphi := \mathbf{F}_\delta \psi \wedge \mathbf{G}_{<\delta} \varphi$$

$$\varphi \ \mathbf{since}_\delta \ \psi := \mathbf{P}_\delta \psi \wedge \mathbf{H}_{<\delta} \varphi$$

where $\mathbf{G}_{<\delta}$ and $\mathbf{H}_{<\delta}$ are the duals of $\mathbf{F}_{<\delta}$ and $\mathbf{P}_{<\delta}$, respectively:

$$\mathbf{G}_{<\delta} \varphi := \neg \mathbf{F}_{<\delta} \neg \varphi$$

$$\mathbf{H}_{<\delta} \varphi := \neg \mathbf{P}_{<\delta} \neg \varphi.$$

Using these metric operators the five quantitative temporal properties of the Introduction can be expressed in the following way:

1. maximal distance: $\mathbf{G}(p \rightarrow \mathbf{F}_{<\delta} q)$
2. exact distance: $\mathbf{G}(p \rightarrow \mathbf{F}_\delta q)$
3. minimal distance: $\mathbf{G}(p \rightarrow \neg \mathbf{F}_{<\delta} p)$
4. periodicity (with period δ): $\mathbf{F}p \wedge \mathbf{G}(p \rightarrow (\neg p \ \mathbf{until}_\delta \ p))$
5. bounded response time: $\exists \delta \ \mathbf{G}(p \rightarrow \mathbf{F}_{<\delta} q)$.

The one but last of these five properties gives periodicity towards the future (the $\mathbf{F}p$ is needed to start the sequence off). Periodicity both towards past and future can be expressed by

$$\mathbf{F}p \wedge \mathbf{G}(p \rightarrow ((\neg p \ \mathbf{until}_\delta \ p) \wedge (\neg p \ \mathbf{since}_\delta \ p))).$$

Note that the definition of $\mathbf{F}_{<\delta}$ above uses quantification over Δ but this is also already essentially needed for the expression of bounded response time (see 5). As stated above, besides constants from Δ (the δ in 1, 2, 3 and 4 above) we incorporate the full first-order language over $(\Delta, +, 0)$. Later on we will also consider a fragment of metric temporal logic in which only constants from Δ are allowed. The formula expressing maximal distance

is strictly stronger than the formula for bounded response time which on its turn is strictly stronger than the formula $\mathbf{G}(p \rightarrow \mathbf{F}q)$ expressing temporal implication in qualitative temporal logic. The latter fact can be illustrated by the following example. Take $T := \mathbb{N}$ with the standard ordering $<$ on $\mathbb{N}$ and the standard distance function ($\Delta = \mathbb{N}$, $d(m, n) = |m - n|$). Let p be true precisely in elements of $\{n(n+1)/2 - 1 \mid n > 0\}$ and q precisely in elements of $\{n(n+1)/2 + n - 1 \mid n > 0\}$. This choice of valuation ensures that the distance between the n-th occurrence of p and the n-th occurrence of q will be n. Therefore, in this metric model bounded response time will fail but qualitative temporal implication still holds.

The ability to quantify over Δ gives metric temporal logic considerable expressive power (see also Theorem 4.1 below). For example, from the metric version of an operator the corresponding qualitative operator can easily be derived by quantifying δ away as we will show below. Furthermore, for qualitative temporal logic the operators **until** and **since** add expressive power (see Section 3) but as just shown their metric versions (and hence by quantifying δ away also their qualitative versions) are expressible in metric temporal logic. Because quantification over Δ contributes significantly to the expressive power of metric temporal logic, we now study the interplay between metric operators and quantification over Δ. We start with the simple case of two existential quantifications

$$\begin{aligned}\mathfrak{M}, t \vDash \exists\delta\, \mathbf{F}_\delta\, \varphi &\equiv \exists t' \in T\, \exists\delta[t < t' \text{ and } d(t, t') = \delta \text{ and } \mathfrak{M}, t' \vDash \varphi] \\ &\equiv \exists t' \in T[t < t' \text{ and } \mathfrak{M}, t' \vDash \varphi] \equiv \mathfrak{M}, t \vDash \mathbf{F}\,\varphi,\end{aligned}$$

so $\exists\delta\, \mathbf{F}_\delta \equiv \mathbf{F}$. By duality also $\forall\delta\, \mathbf{G}_\delta \equiv \mathbf{G}$. In a similar way $\exists\delta\, \mathbf{P}_\delta \equiv \mathbf{P}$ and $\forall\delta\, \mathbf{H}_\delta \equiv \mathbf{H}$. The presence of two identical (either existential or universal) quantifiers is in itself not a sufficient explanation for these equivalences. For example, for classical temporal logic $\mathbf{HG}\varphi \equiv \mathbf{GH}\varphi$ is not valid because of the shifting of the reference point (consider, e.g., $\mathbb{N}$ in the point 0). In the present case, however, identical quantifications over the metric domain and over the set of moments do not influence each other and hence can be interchanged.

More interesting are the cases of alternating quantifiers:

$$\mathfrak{M}, t \vDash \forall\delta\, \mathbf{F}_\delta\, \varphi \equiv \forall\delta\, \exists t' \in T[t < t' \text{ and } d(t, t') = \delta \text{ and } \mathfrak{M}, t' \vDash \varphi].$$

For metric point structures $<$ is comparable and (d1) and (d3) hold. As we have seen above this implies that $\mathbf{F}_\delta$ and $\mathbf{G}_\delta$ are related by

$$\mathbf{F}_\delta\, \varphi \equiv \mathbf{F}_\delta\, \top \wedge \mathbf{G}_\delta\, \varphi.$$

So, when universally quantifying over δ (excluding $\delta = 0$ because $\mathbf{F}_0\, \varphi \equiv \bot$ for all φ) we get

$$\forall\delta[0 \ll \delta \rightarrow \mathbf{F}_\delta\, \varphi] \equiv \forall\delta[0 \ll \delta \rightarrow \mathbf{F}_\delta\, \top] \wedge \mathbf{G}\, \varphi$$

(since $\forall\delta[0 \ll \delta \rightarrow \mathbf{G}_\delta\, \varphi] \equiv \mathbf{G}\, \varphi$). Dually we have

$$\exists\delta[0 \ll \delta \wedge \mathbf{G}_\delta\, \varphi] \equiv \exists\delta[0 \ll \delta \wedge \mathbf{G}_\delta\, \bot] \vee \mathbf{F}\, \varphi.$$

Note that $\forall\delta[0 \ll \delta \Rightarrow \mathfrak{M}, t \vDash \mathbf{F}_\delta \top]$ expresses the requirement that there exists for each $\delta \neq 0$ a point in the future with distance δ from t which is like surjectivity of d but now demanded locally (for t).

Apart from the metric operators $\mathbf{G}_t$, $\mathbf{F}_t$, $\mathbf{H}_t$, and $\mathbf{P}_t$ defined above, we need two additional operators for the proof system to be introduced below. The semantics of these two operators $\mathbf{D}_t$ and $\mathbf{E}_t$ is given by

$$\mathfrak{M}, t \vDash \mathbf{D}_\delta \varphi := \exists t' \in T\,[t' \neq t \text{ and } d(t, t') = \delta \text{ and } \mathfrak{M}, t' \vDash \varphi]$$

$$\mathfrak{M}, t \vDash \mathbf{E}_\delta \varphi := \exists t' \in T\,[d(t, t') = \delta \text{ and } \mathfrak{M}, t' \vDash \varphi].$$

For metric point structures these two operators can be expressed in $\mathbf{F}_t$ and $\mathbf{P}_t$, for example $\mathbf{D}_t := \mathbf{P}_t \vee \mathbf{F}_t$. Nevertheless these new operators are useful by their ability to express the requirements on the distance function d in an independent fashion. To show this we first introduce also the qualitative versions $\mathbf{D}$ and $\mathbf{E}$ of the new metric operators $\mathbf{D}_t$ and $\mathbf{E}_t$ in the same way as was done above for the temporal operators $\mathbf{F}$ and $\mathbf{P}$ with respect to $\mathbf{F}_t$ and $\mathbf{P}_t$:

$$\mathbf{D}\varphi := \exists\delta\, \mathbf{D}_\delta \varphi \text{ and } \mathbf{E}\varphi := \exists\delta\, \mathbf{E}_\delta \varphi.$$

Now all conditions in the definition of a metric point structure can easily be expressed as follows:

(i) totality of $<$ is already expressible in temporal logic with only the qualitative operators $\mathbf{F}$, $\mathbf{P}$ and $\mathbf{D}$:
  (a) transitivity of $<$, i.e., $\forall xyz(x < y < z \to x < z)$, is definable by the formula $\mathbf{FF}\varphi \to \mathbf{F}\varphi$,
  (b) irreflexivity of $<$, i.e., $\forall x \neg x < x$, is definable by the formula $\mathbf{F}\varphi \to \mathbf{D}\varphi$,
  (c) comparability of $<$, i.e., $\forall xy(x < y \vee x = y \vee y < x)$, is definable by the formula $\mathbf{D}\varphi \to (\mathbf{P}\varphi \vee \mathbf{F}\varphi)$

  (For irreflexivity and comparability the additional operator $\mathbf{D}$ next to $\mathbf{F}$ and $\mathbf{P}$ of classical temporal logic is essential because it can be shown that these two operators alone can express neither of these two conditions. The interested reader may consult Chapter 4 of (Koymans 1989) for more information about the powerful extension of classical temporal logic with the $\mathbf{D}$-operator.)

(ii) d surjective: $\forall\delta\, \mathbf{E}\, \mathbf{E}_\delta \top$
  (d1): $\varphi \leftrightarrow \mathbf{E}_0 \varphi$
  (d2): $\forall\delta[(\varphi \wedge \mathbf{E}_\delta \psi) \to \mathbf{E}_\delta(\psi \wedge \mathbf{E}_\delta \varphi)]$
  (d3): $\forall\delta\, \forall\delta'[(\mathbf{F}_\delta\, \mathbf{F}_{\delta'} \varphi \to \mathbf{E}_{\delta+\delta'} \varphi) \wedge (\mathbf{P}_\delta\, \mathbf{P}_{\delta'} \varphi \to \mathbf{E}_{\delta+\delta'} \varphi)]$

(iii) (Δ1)-(Δ6) can be directly formulated in terms of $+$, 0 and quantification over Δ.

To give an example of the four equivalences in (ii) we prove the first one.

First suppose d is surjective. This means that for all $\delta \in \Delta$ there exist $t, t' \in T$ such that $d(t, t') = \delta$. Hence t verifies $\mathbf{E}_\delta \top$. Thus, $\forall\delta\, \mathbf{E}\, \mathbf{E}_\delta \top$ is true.

Conversely, suppose $\forall \delta \, \mathbf{E} \, \mathbf{E}_\delta \top$ is true. Then for all $\delta \in \Delta$ there exists a $t \in T$ such that $\mathbf{E}_\delta \top$ is true in t, implying the existence of a $t' \in T$ at distance δ from t. Thus, $d(t, t') = \delta$, so d is surjective.

Above we showed how the powerful qualitative temporal operators **until** and **since** can be defined in metric temporal logic. The great expressive power of metric temporal logic is perhaps most clearly illustrated by the following theorem.

THEOREM 4.1. All first-order sentences over linear orders are definable in metric temporal logic.

Proof. The main problem in translating first-order conditions on $<$ into equivalent temporal formulas is caused by the possibility to compare in the first-order condition a new variable (corresponding to a more recent reference point in time) with much older variables such as the comparisons between z and x and between u and y in the example

$$\forall x \, \exists y > x \, \exists z < x \, \forall u(z < u < y \rightarrow u = x).$$

Qualitative temporal logics only allow a comparison between a new reference point in time and the most recent reference point before that. Our remedy against this difficulty works as follows. Let $\alpha(x_1, \ldots, x_n)$ be the first-order sentence (containing the bound variables $x_1, \ldots, x_n$) to be defined by a formula from metric temporal logic. First rewrite α in such a way that it only contains the atomic formulas $x_i < x_j$ and $x_i = x_j$ (for $1 \leq i, j \leq n$) and operators $\neg$, $\wedge$ and $\exists$. Furthermore, take care that each atomic formula in the scope of $\exists x_i$ indeed contains x_i (otherwise get the atomic formula outside this scope). The translation of the resulting first-order sentence into a formula from metric temporal logic is based on the following idea. For metric point structures, the comparison of different reference points in time can be accomplished by using the distance function as follows. All points in time are compared with a fixed reference point in time which is characterized by the fact that this point is the only point at which a certain proposition p holds (see below how this can be achieved). The first-order variables $x_1, \ldots, x_n$ are translated into variables $\delta_1, \ldots, \delta_n$ which represent the distance to the fixed reference point (where p holds) taking into account comparisons with other variables using $<$ and $>$ by the appropriate future and past metric operators. The translation of α into a formula from metric temporal logic is given by

$$\mathbf{A}((p \wedge \neg \, \mathbf{D}p) \rightarrow \mu_{(0,\ldots,0)}(\alpha))$$

where $\mathbf{A}$ is the dual of $\mathbf{E}$ ($\mathbf{A}\varphi := \neg \, \mathbf{E} \, \neg \, \varphi$) and the procedure $\mu_{(s_1,\ldots,s_n)}$ is defined below. The prefix $\mathbf{A}((p \wedge \neg \mathbf{D}p) \rightarrow$ fixes the only point where p holds, the fixed reference point. To indicate the comparisons with the fixed reference point the procedure μ uses additional variables $s_1, \ldots, s_n \in \{-, 0, +\}$ ($-$ indicates the past, 0 the present and $+$ the future). Initially $s_1, \ldots, s_n$ are all 0. μ is defined recursively as follows:

$$\mu_{\vec{s}}(\neg\alpha) := \neg\mu_{\vec{s}}(\alpha)$$

$$\mu_{\vec{s}}(\alpha \wedge \beta) := \mu_{\vec{s}}(\alpha) \wedge \mu_{\vec{s}}(\beta)$$

$$\begin{aligned}\mu_{\vec{s}}(\exists x_i\ \alpha) := \exists\delta_i\ \mathbf{E}[&(\delta_i = 0 \wedge p \wedge \mu_{\vec{s}\,[0/s_i]}(\alpha))\\ &\vee (0 \ll \delta_i \wedge \mathbf{F}_{\delta_i} p \wedge \mu_{\vec{s}\,[-/s_i]}(\alpha))\\ &\vee (0 \ll \delta_i \wedge \mathbf{P}_{\delta_i} p \wedge \mu_{\vec{s}\,[+/s_i]}(\alpha))\end{aligned}$$

$$\mu_{\vec{s}}(x_i < x_i) := \bot$$

$$\mu_{\vec{s}}(x_i = x_i) := \top$$

$$\mu_{\vec{s}}(x_i < x_j) := \begin{cases}\mathbf{F}p & \text{if } s_j = 0\\ \mathbf{F}\mathbf{F}_{\delta_j} p & \text{if } s_j = -\\ \mathbf{F}\mathbf{P}_{\delta_j} p & \text{if } s_j = +\end{cases}$$

$$\mu_{\vec{s}}(x_j < x_i) := \begin{cases}\mathbf{P}p & \text{if } s_j = 0\\ \mathbf{P}\mathbf{F}_{\delta_j} p & \text{if } s_j = -\\ \mathbf{P}\mathbf{P}_{\delta_j} p & \text{if } s_j = +\end{cases}$$

$$\mu_{\vec{s}}(x_i = x_j) := \begin{cases}p & \text{if } s_j = 0\\ \mathbf{F}_{\delta_j} p & \text{if } s_j = -\\ \mathbf{P}_{\delta_j} p & \text{if } s_j = +\end{cases}$$

where x_i in the last five cases (from $x_i < x_i$ onwards) is the bound variable belonging to the smallest enclosing existential quantification.

Finally, we look at axiomatizations for metric temporal logic. Completeness may be unattainable because of the very powerful quantification over Δ. By assuming an oracle for Δ *relative* completeness results might be obtained, however. Instead of attempting to axiomatize metric temporal logic completely we can at least provide a sound axiomatization:

DEFINITION 4.2. The metric temporal logic proof system M consists of

0. the definitions

$$\begin{aligned}\exists\delta\ \varphi(\delta) &:= \neg\ \forall\delta\ \neg\ \varphi(\delta),\\ \mathbf{G}_t\ \varphi &:= \neg\ \mathbf{F}_t\ \neg\ \varphi,\\ \mathbf{H}_t\ \varphi &:= \neg\ \mathbf{P}_t\ \neg\ \varphi,\\ \mathbf{F}\ \varphi &:= \exists\delta\ \mathbf{F}_\delta\ \varphi,\\ \mathbf{G}\ \varphi &:= \forall\delta\ \mathbf{G}_\delta\ \varphi,\\ \mathbf{P}\ \varphi &:= \exists\delta\ \mathbf{P}_\delta\ \varphi,\\ \mathbf{H}\ \varphi &:= \forall\delta\ \mathbf{H}_\delta\ \varphi,\end{aligned}$$

1. A complete axiomatization of predicate logic including MP (Modus Ponens) and the following two rules ($\forall$-elimination, respectively $\forall$-introduction):
   a. to infer $\varphi(t)$ from $\forall\delta\ \varphi(\delta)$, where $\varphi(t)$ is the result of substituting the term t from the first-order structure $(\Delta, +, 0)$ properly (i.e., avoiding that any free variable of t becomes bound) for all occurrences of δ in $\varphi(\delta)$,

b. to infer $\varphi \to \forall\delta\, \psi(\delta)$ from $\varphi \to \psi(t)$, where t is a term from the first-order structure $(\Delta, +, 0)$ that does not appear in $\varphi \to \forall\delta\, \psi(\delta)$,

2. The distribution axiom schemas and temporalization rules of the standard proof system for classical temporal logic (see (Benthem 1983)), but now for $\mathbf{G}_t$ and $\mathbf{H}_t$ instead of $\mathbf{G}$ and $\mathbf{H}$:
   a. $\mathbf{G}_t(\varphi \to \psi) \to (\mathbf{G}_t\, \varphi \to \mathbf{G}_t\, \psi)$ and
   $\mathbf{H}_t(\varphi \to \psi) \to (\mathbf{H}_t\, \varphi \to \mathbf{H}_t\, \psi)$,
   b. to infer $\mathbf{G}_t\, \varphi$ from φ, and to infer $\mathbf{H}_t\, \varphi$ from φ,
3. the characterizations (i)–(iii) of the properties of a metric point structure above,
4. the already mentioned additional relationships between metric operators:

$$\mathbf{F}_t\, \varphi \leftrightarrow \mathbf{F}_t\, \top \wedge \mathbf{G}_t\, \varphi,$$
$$\mathbf{P}_t\, \varphi \leftrightarrow \mathbf{P}_t\, \top \wedge \mathbf{H}_t\, \varphi,$$

5. Axiom schemas relating to arithmetic over the metric domain:
   a. $\mathbf{F}_0\, \varphi \leftrightarrow \bot \leftrightarrow \mathbf{P}_0\, \varphi$
   b. $\mathbf{F}_{t_1}\, \dot{\mathbf{F}}_{t_2}\, \varphi \leftrightarrow \mathbf{F}_{t_1}\, \top \wedge \mathbf{F}_{t_1+t_2}\, \varphi$,
   $\mathbf{P}_{t_1}\, \dot{\mathbf{P}}_{t_2}\, \varphi \leftrightarrow \mathbf{P}_{t_1}\, \top \wedge \mathbf{P}_{t_1+t_2}\, \varphi$,
   where $\dot{\mathbf{F}}_t$ and $\dot{\mathbf{P}}_t$ are the reflexive closure of $\mathbf{F}_t$ and $\mathbf{P}_t$, respectively:

$$\dot{\mathbf{F}}_t\, \varphi := (t = 0 \wedge \varphi) \vee \mathbf{F}_t\, \varphi \text{ and } \dot{\mathbf{P}}_t\, \varphi := (t = 0 \wedge \varphi) \vee \mathbf{P}_t\, \varphi.$$

   c. $\mathbf{F}_{t_1}\, \mathbf{P}_{t_1+t_2}\, \varphi \leftrightarrow \mathbf{F}_{t_1}\, \top \wedge \dot{\mathbf{P}}_{t_2}\, \varphi$,
   $\mathbf{P}_{t_1}\, \mathbf{F}_{t_1+t_2}\, \varphi \leftrightarrow \mathbf{P}_{t_1}\, \top \wedge \dot{\mathbf{F}}_{t_2}\, \varphi$,
   $\mathbf{F}_{t_1+t_2}\, \mathbf{P}_{t_1}\, \varphi \leftrightarrow \mathbf{F}_{t_1+t_2}\, \top \wedge \dot{\mathbf{F}}_{t_2}\, \varphi$,
   $\mathbf{P}_{t_1+t_2}\, \mathbf{F}_{t_1}\, \varphi \leftrightarrow \mathbf{P}_{t_1+t_2}\, \top \wedge \dot{\mathbf{P}}_{t_2}\, \varphi$.

From this proof system several interesting properties can be derived such as

(1) $\forall\delta\, \mathbf{G}\, \varphi(\delta) \leftrightarrow \mathbf{G}\forall\delta\, \varphi(\delta)$ and $\mathbf{F}\forall\delta\, \varphi(\delta) \to \forall\delta\, \mathbf{F}\, \varphi(\delta)$ (these follow from predicate logic and the definitions $\mathbf{G}\varphi \equiv \forall\delta'\, \mathbf{G}_{\delta'}\, \varphi$ and $\mathbf{F}\varphi \equiv \exists\delta'\, \mathbf{F}_{\delta'}\, \varphi$),
(2) $\forall\delta\, \mathbf{F}_\delta\, \varphi \leftrightarrow \forall\delta\, \mathbf{F}_\delta\, \top \wedge \mathbf{G}\, \varphi$ by predicate logic and clause 4 in the definition of M above,
(3) $\mathbf{F}_t\, \mathbf{P}_t\, \varphi \leftrightarrow \mathbf{F}_t\, \top \wedge \varphi$ by taking $t_2 = 0$ in the first axiom schema of clause 5c in the definition of M above

and similarly for the mirror images (obtained by exchanging $\mathbf{G}$ with $\mathbf{H}$ and $\mathbf{F}$ with $\mathbf{P}$).

The next properties are important enough to derive them as theorems of M. In these derivations MP abbreviates Modus Ponens and M followed by a number indicates the corresponding clause in the definition of M above.

PROPOSITION 4.1. $\vdash_M \mathbf{G}_t(\varphi \wedge \psi) \leftrightarrow \mathbf{G}_t\, \varphi \wedge \mathbf{G}_t\, \psi$

Proof. This theorem of M can be derived as follows.

1. $\varphi \wedge \psi \to \varphi$	(propositional logic)
2. $\mathbf{G}_t(\varphi \wedge \psi \to \varphi)$	(1, M2b)
3. $\mathbf{G}_t(\varphi \wedge \psi) \to \mathbf{G}_t\varphi$	(2, MP, M2a)

4. $\mathbf{G}_t(\varphi \wedge \psi) \rightarrow \mathbf{G}_t\psi$	(analogous to 1–3)
5. $\mathbf{G}_t(\varphi \wedge \psi) \rightarrow \mathbf{G}_t\varphi \wedge \mathbf{G}_t\psi$	(3, 4)
6. $\varphi \rightarrow (\psi \rightarrow \varphi \wedge \psi)$	(propositional logic)
7. $\mathbf{G}_t(\varphi \rightarrow (\psi \rightarrow \varphi \wedge \psi))$	(6, *M*2b)
8. $\mathbf{G}_t\,\varphi \rightarrow \mathbf{G}_t(\psi \rightarrow \varphi \wedge \psi)$	(7, MP, *M*2a)
9. $\mathbf{G}_t\,\varphi \rightarrow (\mathbf{G}_t\,\psi \rightarrow \mathbf{G}_t(\varphi \wedge \psi))$	(8, MP, *M*2a)
10. $\mathbf{G}_t\,\varphi \wedge \mathbf{G}_t\,\psi \rightarrow \mathbf{G}_t(\varphi \wedge \psi)$	(9, propositional logic)
11. $\mathbf{G}_t(\varphi \wedge \psi) \leftrightarrow \mathbf{G}_t\,\varphi \wedge \mathbf{G}_t\,\psi$	(5, 10)

This was not very surprising since this holds also for the nonmetric case: $\mathbf{G}(\varphi \wedge \psi) \leftrightarrow \mathbf{G}\varphi \wedge \mathbf{G}\psi$ (and indeed the derivation above uses only clause 2 of *M* which stems from the standard proof system for classical temporal logic). However, in contrast with the nonmetric case we have also the following:

PROPOSITION 4.2. $\vdash_M \mathbf{F}_t(\varphi \wedge \psi) \leftrightarrow \mathbf{F}_t\,\varphi \wedge \mathbf{F}_t\,\psi$

Proof. This theorem of *M* can be derived as follows.

1. $\mathbf{F}_t(\varphi \wedge \psi) \leftrightarrow \mathbf{F}_t\,\top \wedge \mathbf{G}_t(\varphi \wedge \psi)$	(*M*4)
2. $\mathbf{G}_t(\varphi \wedge \psi) \leftrightarrow \mathbf{G}_t\,\varphi \wedge \mathbf{G}_t\,\psi$	(Proposition 4.1)
3. $\mathbf{F}_t(\varphi \wedge \psi) \leftrightarrow \mathbf{F}_t\,\top \wedge \mathbf{G}_t\,\varphi \wedge \mathbf{G}_t\,\psi$	(1, 2)
4. $\mathbf{F}_t\,\top \wedge \mathbf{G}_t\,\varphi \leftrightarrow \mathbf{F}_t\,\varphi$	(*M*4)
5. $\mathbf{F}_t\,\top \wedge \mathbf{G}_t\,\psi \leftrightarrow \mathbf{F}_t\,\psi$	(*M*4)
6. $\mathbf{F}_t(\varphi \wedge \psi) \leftrightarrow \mathbf{F}_t\,\varphi \wedge \mathbf{F}_t\,\psi$	(3, 4, 5)

The only part of the standard proof system for classical temporal logic that we did not take over concerns the tense mixing axiom schemas $\varphi \rightarrow \mathbf{GP}\varphi$ and $\varphi \rightarrow \mathbf{HF}\varphi$. These are however theorems of *M*, for example, the first one:

PROPOSITION 4.3. $\vdash_M \varphi \rightarrow \mathbf{GP}\varphi$

Proof. This theorem of *M* can be derived as follows.

1. $\mathbf{F}_t\,\top \vee \neg\,\mathbf{F}_t\,\top$	(propositional logic)
2. $\mathbf{F}_t\,\top \wedge \varphi \rightarrow \mathbf{F}_t\,\mathbf{P}_t\,\varphi$	(*M*5c)
3. $\mathbf{F}_t\,\mathbf{P}_t\,\varphi \leftrightarrow \mathbf{F}_t\,\top \wedge \mathbf{G}_t\,\mathbf{P}_t\,\varphi$	(*M*4)
4. $\mathbf{F}_t\,\top \wedge \varphi \rightarrow \mathbf{G}_t\,\mathbf{P}_t\,\varphi$	(2, 3)
5. $\neg\,\mathbf{F}_t\,\neg\,\mathbf{P}_t\,\varphi \leftrightarrow \neg\,(\mathbf{F}_t\,\top \wedge \mathbf{G}_t\,\neg\,\mathbf{P}_t\,\varphi)$	(*M*4)
6. $\neg\,\mathbf{F}_t\,\top \rightarrow \neg\,\mathbf{F}_t\,\neg\,\mathbf{P}_t\,\varphi$	(5)
7. $\neg\,\mathbf{F}_t\,\top \rightarrow \mathbf{G}_t\,\mathbf{P}_t\,\varphi$	(6, *M*0)
8. $\varphi \rightarrow \mathbf{G}_t\,\mathbf{P}_t\,\varphi$	(1, 4, 7)
9. $\varphi \rightarrow \forall\delta\,\mathbf{G}_\delta\,\mathbf{P}_\delta\,\varphi$	(8, *M*1b)
10. $\varphi \rightarrow \forall\delta\,\mathbf{G}_\delta\,\exists\delta'\,\mathbf{P}_{\delta'}\,\varphi$	(9, $\delta' = \delta$)
11. $\varphi \rightarrow \mathbf{GP}\varphi$	(10, *M*0)

Another possibility is to eliminate the quantification over Δ by only allowing constants from Δ. Such a fragment of metric temporal logic could be based on the following eight temporal operators: $\mathbf{until}_{<\delta}$, $\mathbf{until}_{\delta}$, $\mathbf{until}_{>\delta}$, **until**, $\mathbf{since}_{<\delta}$, $\mathbf{since}_{\delta}$, $\mathbf{since}_{>\delta}$, **since** where δ may be any constant from Δ. Notice that we now included the qualitative operators **until** and **since** because these can no longer be obtained by quantification over their metric equivalents. An alternative way to look at these qualitative operators is to see them as special metric operators $\mathbf{until}_{<\infty}$ and $\mathbf{since}_{<\infty}$ as is done in (Hooman and Widom 1989). In this view ∞ is not an element of Δ but it is added to $\ll$ as its greatest element. (Hooman and Widom 1989) is also interesting for another reason: it shows how metric temporal logic can be embedded as an assertion language into a compositional proof system.

Another look at the constants δ from Δ is to consider them as programs from a kind of dynamic logic (see (Harel 1984)) by defining

$$[\delta] := \{(t, t') \mid d(t, t') = \delta\}$$

with the following additional program structure

0: the 'skip' program
+: sequential composition;

and the property that all programs are *deterministic*:

$$\mathbf{F}_{\delta}\, \varphi \wedge \mathbf{F}_{\delta}\, \psi \rightarrow \mathbf{F}_{\delta}\, (\varphi \wedge \psi)$$

(cf. Proposition 4.2). This connection with dynamic logic deserves further investigation.

In the same way as indicated in Section 3 for L(**until**, **since**) we can introduce global variables and quantification over them in order to reason about (possibly infinite) data domains. This will be illustrated in the next section.

5. Specification examples

In this section we illustrate the application of metric temporal logic to real-time systems by a series of examples. The first three examples treat some simple, but characteristic, pure real-time phenomena: pure time-out, a watchdog timer monitoring a processor and the wait/delay statement. The next four examples combine features of message passing and real-time systems. Example four concerns a terminal adaptor where the speed of the incoming data is higher than the speed of the outgoing data. In example five, a synchronous and an asynchronous input are mixed into one synchronous output. Example six treats an abstract transmission medium. Real-time communication constructs like send and receive with time-out are the subject of example seven. The last example deals with process control systems in which continuously changing entities play an important role.

The priority of operators in the specification examples is as follows: unary operators have the highest priority followed by **until** and **since**-like operators (including the **unless**-operator defined below), then come $\wedge$ (conjunction) and $\vee$ (disjunction) and the least priority

is given to $\rightarrow$ (implication) and $\leftrightarrow$ (equivalence). With respect to priority, universal and existential quantification are treated as unary operators.

In our specifications we assume not only linearity of the ordering but also that the ordering is unbounded towards the future in order to reason about infinite behavior. This involves the qualitative part of metric temporal logic. For the quantitative part we assume local surjectivity of the temporal distance function d, that is, we assume $\forall\delta\,[0 \ll \delta \rightarrow \mathbf{F}_\delta\,\top]$. An important consequence of this is $\mathbf{F}_\delta \equiv \mathbf{G}_\delta$ for all $\delta \neq 0$ since $\mathbf{F}_\delta\,\varphi \equiv \mathbf{F}_\delta\,\top \wedge \mathbf{G}_\delta\,\varphi$ (see Section 4). The standard metric point structures that we have in mind use respectively the natural numbers, the integers, the (non-negative) rational numbers and the (non-negative) real numbers for the time domain T and the non-negative part of T for Δ where $<$, $+$ and 0 have the standard interpretation for these number systems and d is the absolute difference. For example, one of the standard metric point structures is

$$(\mathsf{ZZ},\ \mathsf{IN},\ <,\ d,\ +,\ 0)$$

where $<$ is the standard ordering on ZZ, $+$ the standard addition on IN, 0 the standard constant from IN and d is defined by

$$d(z, z') := |z - z'|.$$

In the specifications, we leave out universal quantifications over the data domains (so all free variables ranging over a data domain should be universally quantified by a series of universal quantifiers in front of the given axiom).

For the specification examples in this section we need several additional qualitative temporal operators. First we need the reflexive counterparts of **P** and **F**:

$$\dot{\mathbf{P}}\,\varphi := \varphi \vee \mathbf{P}\,\varphi$$

and

$$\dot{\mathbf{F}}\,\varphi := \varphi \vee \mathbf{F}\,\varphi.$$

Instead of $\dot{\mathbf{F}}$ we will use the more usual representation in computer science $\Diamond$ (see Section 3). Next we need a reflexive version of **since** which we will denote by **since**. Semantically it corresponds to replacing every $<$ in the definition of **since** by $\leq$. Syntactically this can be achieved by the definition

$$\varphi_1 \ \underline{\textbf{since}}\ \varphi_2 := (\varphi_1 \wedge \varphi_2) \vee (\varphi_1 \wedge \varphi_1\ \textbf{since}\ (\varphi_1 \wedge \varphi_2)).$$

Finally we also need a unary operator denoted by **J** representing that its argument has *just* become true:

$$\mathbf{J}\varphi := \varphi \wedge (\mathbf{P}\varphi \rightarrow \neg\ \varphi\ \textbf{since}\ \neg\ \varphi).$$

(We thank Job Zwiers for the discussion leading to this more concise representation of this operator than the one contained in (Koymans 1989).) This definition can be explained

as follows. Apart from the obvious first part demanding that φ holds at the current moment, this formula describes that there was a period immediately before (how small it may be) such that φ was false in that period. Note that for a formula φ that is true on the rationals and false on the irrational numbers $\mathbf{J}\varphi$ is never true (this corresponds to our intuition that φ changes its truth value infinitely fast and hence cannot have become *just* true).

In our examples we will encounter periodicity requirements. Unconditional periodicity of an event e with period δ can be formulated by

$$\mathit{periodic}(e, \delta) := e \rightarrow \neg\, e\ \mathbf{until}_{\delta}\ e.$$

Furthermore, conditional periodicity can be defined by adding a condition c to the antecedent:

$$\mathit{periodic}(e, \delta, c) := e \wedge c \rightarrow \neg\, e\ \mathbf{until}_{\delta}\ e.$$

In applying metric temporal logic to practical examples the metric domain Δ should be associated with a time unit relevant for that application, usually second or a derivative thereof. However, in principle other time units, such as number of shaft rotations, are allowed too. Connected with this is the translation of elements of data domains that represent time units into elements of Δ. We will represent this translation by a function δ. For example, when the data domain represents milliseconds and Δ counts in seconds then we can take

$$\delta(t) = \frac{t}{1000}.$$

In case the data domain has more structure, one may want to impose additional conditions on δ, for example, when the data domain is ordered monotonicity of δ with respect to this ordering and when the data domain incorporates addition distributivity of δ with respect to this addition. The most simple case occurs when the data domain can be embedded in the metric domain. In such a case it suffices to take for δ simply the embedding mapping.

Since we will combine message passing and real-time features in the last four examples, we now briefly indicate how message passing systems can be specified with temporal logic (see Chapter 5 of (Koymans 1989) for more details and examples). In the following we are only interested in describing the external behavior of message passing systems. Therefore, we view such a system as a black box. A message passing system, then, is a system that gets messages and passes these messages on to their destination. If we denote the input of a message m by $in(m)$ and the delivery of a message m by $out(m)$, Figure 2 represents a message passing system as a black box. Here, the source and destination of a message are left implicit, that is, $in(m)$ means that there is a source that gives m to the message

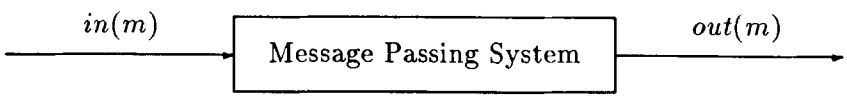


Figure 2. Message passing system as a black box.

passing system and *out*(*m*) means that the message passing system delivers *m* to its destination (note the asymmetry: the destination of a message must always be known, while this is not necessarily the case for the source). When sources and destinations are explicitly represented we get *in*(*s*, *m*) and for symmetry reasons *out*(*d*, *m*) where, however, always $d = destination(m)$.

The external behavior of a message passing system is characterized by its input sequence, its output sequence and their relation in time. Hence, only input, output and their relation determine the observable difference between several types of message passing systems. This means that quite different message passing systems such as a simple buffer (or transmission medium) and a complex communication network should be considered the same as long as they exhibit the same observable (external) behavior, that is, the same relation in time between input and output.

The following basic assumption about *in*, *out* and their relation in time is characteristic for all message passing systems:

NC the message passing system does not create messages by itself neither

NC1 by creating new messages (a message is new when it has not been given to the message passing system before), nor

NC2 by delivering duplicates of messages given to the message passing system.

In other words: the bag of delivered messages is always *some part* of the bag of messages that have been given to the message passing system. **NC** is an abbreviation for No Creation. All message passing systems are required to satisfy this assumption because they are intended to *pass* messages and not to modify/create or replicate messages. Although it is known that neither **NC1** nor **NC2** can be guaranteed completely in practice, it makes sense to make such slightly idealized assumptions. Anyway one always has the option of dropping one or both of them (although in case of dropping **NC1** this would allow the system to exhibit almost any behavior). **NC** is the basic safety assumption for message passing systems in the sense that the system does not commit a bad thing (see, e.g., (Lamport 1983a) for this characterization of safety) by creating messages. Concerning liveness, the basic assumption is that at least some messages that have been given to the system will be delivered at their destination, as formulated in the following liveness assumption:

LA if an infinite number of messages will be given to the message passing system, an infinite number of these will be delivered at their destination.

Stated informally, the system may lose an arbitrary number of messages in a row, but eventually it should deliver at least one message (and since time extends to infinity repeating this we get the delivery of a second message, a third message et cetera).

In the above representation of message passing systems we assume that both *in*(*m*) and *out*(*m*) cause no blocking, that is, the message passing system can never refuse a message that is given to it (it always accepts the message) and it is always able to deliver a message to its destination. In practice, this is usually achieved by associating input and output queues at both ends of the message passing system (if we do not make the unrealistic assumption of infinite queues, this implies that *in*(*m*) leads to the loss of *m* when the input queue is full and similarly for *out*(*m*) and the output queue).

Because of the physical limitations in the real world it makes sense to make also the following assumption of finite speed:

FS the speed of the message passing system is finite, that is, there is a positive (infinite in case the message gets lost) delay between the acceptance of a message and its delivery.

As is apparent from the picture above, the interface between the message passing system and its environment consists of *in* and *out*. Sometimes more information about the interface is available, for example that there is only a single input line or a single output line (a line is called single when at any time there can be at most one message present on the line) leading to the following assumptions no simultaneous input and no simultaneous ouptut:

SI at any moment of time, at most one message can be given to the system,
SO at any moment of time, at most one message can be delivered to its destination.

These assumptions apply in particular to the case of a single source and a single destination or in case of explicit representation of sources and destinations for each source and destination separately. Although there cannot be two messages at the same time given to the system nor delivered by the system, it is perfectly possible that there is a message given to the system simultaneously with the delivery of a (different) message by the system. Apart from the assumptions **SI** and **SO** being enforced by the interface it is also possible that the environment, respectively the system, will ensure that no simultaneous inputs, respectively outputs, occur (in spite of the presence of several input, respectively output, lines). This is the reason that the nomenclature single input and single output would be misleading for the above assumptions **SI** and **SO**; therefore we call them no simultaneous input and no simultaneous output, respectively.

In the above description it is not stated whether $in(m)$ and $out(m)$ are considered to be instantaneous or to have a certain duration. Anyway, for message passing systems it can be assumed that they are instantaneous, because it is always possible to identify a unique moment of time at which a message can be said to be accepted, respectively delivered: Take for example the case where a message consists of bytes, then one can let $in(m)$ and $out(m)$ correspond to the input (respectively output) of the last byte of m (since we assumed that bytes are not observable but only messages, $in(m)$ can be seen as instantaneous, although on a finer level of granularity the different bytes can be seen).

An example of a message passing sytem often occurring in practice that is subject to the above restrictions (**NC**, **LA**, **FS**, **SI**, **SO**) is a transmission medium with a probability between zero and one of a successful transmission. Such a message passing system exhibits only external behaviors that are allowed by these restrictions although the probability of the occurrence of certain behaviors may vary.

Apart from the above restrictions, message passing systems can be distinguished by requiring additional properties. As we saw above the basic liveness requirement for a message passing system is that at least some of the accepted messages will be delivered. Sometimes we need the stronger requirement that *all* accepted messages will eventually be delivered in which case we will call the system perfect. In case messages may get lost (an imperfect

system) this notion of a lost message must again be considered as a purely external one, that is, whenever an accepted message is never delivered it is considered as being lost, although it may remain forever in the message passing system (and is not lost in the internal view of that system; an example is a network with a routing algorithm that does not guarantee that each message will eventually reach its destination).

Another distinction can be made by requiring a certain order in which accepted messages are delivered (if at all). In the above we imposed no order at all (this corresponds to a bag-like behavior). As an additional requirement one can pose FIFO ordering (first-in first-out, like queues) or LIFO ordering (last-in first-out, like stacks). It should be noted, however, that the pure data structure view of queues and stacks is complicated by the fact that these can be operated upon in parallel in case of message passing systems by the input and output of messages (for a stack a simultaneous pop and push, for example). An example of a FIFO message passing is an ordinary buffer. An example of an unordered (that is, in no order at all) message passing system is a communication network in which each message is sent on to an intermediate node depending on some routing algorithm. Due to, for example, congestion on the chosen route, later messages may arrive earlier when sent via alternative routes.

It can be proved that a large number of natural classes of message passing systems cannot be specified with strong temporal logic (see Section 5.4 of (Koymans 1989)). The reason for this problem is assumption **NC2** above: In case of duplicated messages it becomes impossible to couple each message that is delivered by a message passing system to a *unique* message accepted by that system. This result seems to necessitate the enrichment of temporal logic for the specification of message passing systems, for example, with auxiliary data structures or history variables. In Section 5.5 of (Koymans 1989) it is shown that no such enrichment is logically required by introducing an additional axiom within temporal logic which formalizes the assumption that messages accepted by the system can be uniquely identified, for example, by means of *conceptual* time stamps. This assumption can be justified by the notion of data-independence of (Wolper 1986). Informally, a system is called data-independent when the values of the supplied data do not influence the functional behavior of the system. Since message passing systems are intended to *pass* data, they can be viewed as being data-independent. One of the results of (Wolper 1986) implies that the correctness of a data-independent system does not depend on the uniqueness of the incoming data. Hence, this assumption of unique identification is not really a restrictive one.

We now show how pure message passing systems can be specified with the temporal logic of Section 3. Because *in* and *out* can be considered to be instantaneous we can model *in* and *out* by (unary) predicates.

First we formulate our assumption about the uniqueness of *incoming* messages (the Unique Identification assumption):

MP1 $\quad in(m) \rightarrow \neg\ \mathbf{P}\ in(m).$

Here and in the sequel, MP is an abbreviation for message passing. Under this Unique Identification assumption the most important basic assumption of message passing systems, No Creation can be specified by:

MP2a $out(m) \rightarrow \dot{\mathbf{P}}\ in(m)$
MP2b $out(m) \rightarrow \neg\ \mathbf{P}\ out(m)$.

The first of these two axioms represents the demand that a message passing system does not create *new* messages, while the second axiom represents the absence of duplicate messages (since the input consists of unique messages by the Unique Identification assumption, the output must also consist of unique messages because no messages may be created by the message passing system). Of course these two axioms can be combined into one:

MP2 $out(m) \rightarrow \dot{\mathbf{P}}\ in(m) \wedge \neg\ \mathbf{P}\ out(m)$.

Notice that the axioms **MP2a** and **MP1** taken together imply that $in(m) \rightarrow \neg\ \mathbf{P}\ out(m)$.

In general, when perfectness of the message passing system is not assumed, the basic liveness assumption **LA** above is essential to ensure that at least *some* messages arrive (otherwise the system that throws all messages away would satisfy all conditions for a message passing system):

MP3 $\mathbf{G}\ \mathbf{F}\ \exists\ m\ in(m) \rightarrow \mathbf{F}\ \exists\ m\ out(m)$.

Above also the assumption **FS** of finite speed is mentioned for realistic purposes. Finite speed can be enforced by replacing the $\dot{\mathbf{P}}$-operator in axiom **MP2a** above by its strict (i.e., irreflexive) version **P** and similarly for axiom **MP2**:

MP2a′ $out(m) \rightarrow \mathbf{P}\ in(m)$
MP2′ $out(m) \rightarrow \mathbf{P}\ in(m) \wedge \neg\ \mathbf{P}\ out(m)$.

No simultaneous input and no simultaneous output can be specified respectively by

MP4a $in(m) \wedge in(m') \rightarrow m' = m$
MP4b $out(m) \wedge out(m') \rightarrow m' = m$.

This concludes the survey of the first set of assumptions for message passing systems. We now turn to the additional assumptions about perfectness and ordering. The perfectness of a message passing system (which implies the basic liveness assumption above) can be expressed by

MP3′ $in(m) \rightarrow \Diamond\ out(m)$.

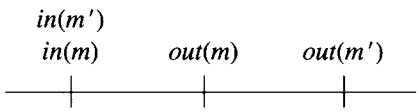
When finite speed is assumed, the $\Diamond$ in the axiom above can be replaced by its strict version **F**. What remains is the specification of special orderings of the output with respect to the input. We look at the cases of FIFO (queue-like) and LIFO (stack-like). First-in first-out requires the same ordering in the output as in the input:

MP5 $out(m) \wedge \mathbf{P}\ out(m') \rightarrow \dot{\mathbf{P}}\ (in(m) \wedge \dot{\mathbf{P}}\ in(m'))$.

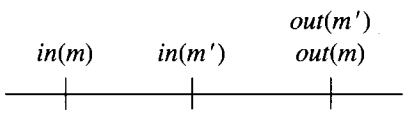
The above axiom suffices when no simultaneous output is assumed. Otherwise also the case when two messages are output at the same time should be considered. This is reflected in the following axiom:

MP5x $out(m) \wedge out(m') \rightarrow \dot{\mathbf{P}}\ (in(m) \wedge in(m'))$.

This exception is caused by the following asymmetry between input and output when requiring FIFO-behavior:



is allowed (when m and m' are input at the same time none of these messages can be said to have come in first, so they may be output in an arbitrary order), but

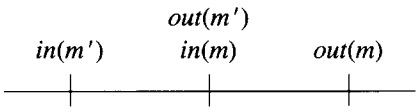


is not (when m is input before m', it should also come out first in the output).

For last-in first-out we get similar specifications (although a bit more complicated) because stack-like behavior allows apart from the reversal of the ordering from output and that from input also the possibility that a message has already been output by the system in the meantime, whence a comparison with a message that has been input after that is not needed any more:

MP6 $out(m) \wedge \mathbf{P}\ out(m') \rightarrow \mathbf{P}\ (in(m') \wedge \dot{\mathbf{P}}\ in(m)) \vee \mathbf{P}\ (out(m') \wedge \neg\ \mathbf{P}\ in(m))$.

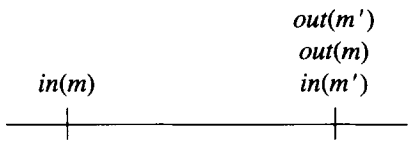
Here we consider



as correct LIFO-behavior (otherwise the last **P** in the axiom above should be replaced by its reflexive version $\dot{\mathbf{P}}$). This is comparable with a simultaneous pop and push (recall that input and output on both sides of our queues and stacks can operate in parallel, for example, the case $in(m) \wedge out(m')$ is always possible, also when assuming no simultaneous input and no simultaneous output). Just as in the FIFO-case, when no simultaneous output is not assumed, an additional axiom is needed, in this case:

MP6x $out(m) \wedge out(m') \rightarrow (\neg (in(m) \vee in(m')) \rightarrow \mathbf{P}\ (in(m) \wedge in(m')))$.

Again there is a little complication, this time because of the correct LIFO-behavior (unless we suppose finite speed):



(although m' comes in last, m can be considered to have been already output). This concludes our sketch of the application of temporal logic to message passing systems.

In examples three and seven we look at statements from concurrent programming languages such as CHILL (Chill 1980) or Ada (Programming Language Ada 1983). For expressing the semantics of programming languages we use location variables l and location predicates *at* and *after*. The first assumption on locations is that being simultaneously at and after the same location is impossible (being simultaneously at different locations in different processes or tasks is of course possible):

L1 $\quad \neg(at(l) \wedge after(l))$.

Locations are special data elements and as such we can impose on them the Unique Identification assumption. However, being present at a certain location is not instantaneous, but has some duration, so the uniqueness of locations is expressed by

L2 $\quad at(l) \rightarrow at(l)$ **unless** $(after(l) \wedge \mathbf{G} \neg at(l))$

where **unless** is the weak version of the **until** which does not require that its second argument will become true eventually:

φ_1 **unless** $\varphi_2 := \mathbf{G}\ \varphi_1 \vee \varphi_1$ **until** φ_2.

5.1. Example 1: Pure time-out

One of the most common and easiest real-time constructs is the time-out. A time-out is generated at the end of a period (whose length is determined by the value by which the timer was set) in which a certain event (think of the signal resetting the timer) has not occurred. Time-outs are widely used in real-time systems to safeguard one part of a system against malfunction of another part. Let the event be e and the time-out value δ, then the time-out on e after δ can be defined in metric temporal logic by

time-out$(e, \delta) := \neg \mathbf{P}_{<\delta}\ e$.

So, a time-out on e after δ is generated if and only if e has not occurred during the last δ time units. Notice that in this representation the setting of the timer is considered irrelevant.

If we want to incorporate this, however, let *set* and *reset* be the event setting, respectively resetting, the timer, then a time-out with period δ can be described by

$$\neg \mathit{reset}\ \mathbf{since}_{\delta}\ \mathit{set}.$$

5.2. *Example 2: Watchdog timer*

This example concerns a pure real-time system, a watchdog timer. A processor is monitored by a timer, the watchdog. The processor sets the timer by a signal *enable*(t) and it should reset the timer by a *reset* signal each time before the timer expires (cf. the previous example). When the processor does not succeed in resetting the timer in time, the processor will be stopped by a *halt* signal from the watchdog. At any time, the processor and the watchdog timer can be restarted by an *initiate* signal from the environment (e.g., an operator pushing a button). After an *initiate* signal a new period of enabling and resetting the timer starts. Once the timer is set with *enable*(t) after an initiate signal, the time-out period cannot be changed (and thus every subsequent *enable*(t') signal is ignored) until the next *initiate* signal. Figure 3 summarizes this state of affairs.

We assume that two settings of the timer cannot occur simultaneously (otherwise the time-out period could be unknown):

$$\mathit{enable}(t) \wedge \mathit{enable}(t') \rightarrow t' = t.$$

To identify the first *enable*(t) after an initiate signal we define

$$\mathit{firstenable}(t) := \mathit{enable}(t) \wedge (\neg\ \exists\ t'\ \mathit{enable}(t'))\ \mathbf{since}\ \mathit{initiate}.$$

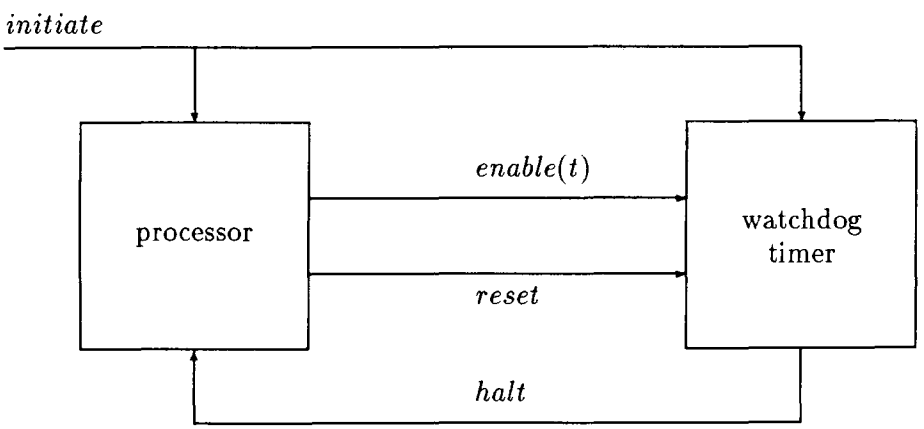


Figure 3. Watchdog timer.

The only essential thing to be specified is the generation of the *halt* signal. This is characterized by a period bounded by *firstenable*(t) (timer set) and a *halt* signal (timer stopped) in which:

1. no *initiate* and no *halt* signal occurred during this whole period (no *halt* signal since we want at most one *halt* signal to occur between two *initiate* signals),
2. no *reset* occurred during the last t time units of this period.

The generation of a *halt* signal can then be specified by a nested since formula:

$$halt \leftrightarrow \exists\, t[t > 0 \wedge (\neg\ initiate \wedge \neg\ halt \wedge \neg\ reset)\ \mathbf{since}_{\delta(t)} \\ ((\neg\ initiate \wedge \neg\ halt)\ \underline{\mathbf{since}}\ firstenable(t))]$$

where δ transfers an element from the data domain of *enable* to an element of the metric domain Δ (see the introduction of this section).

5.3. Example 3: Wait/delay statement

This example treats the wait statement or delay statement as occurring in concurrent programming languages such as CHILL (CHILL 1980) or Ada (Programming Language Ada 1983). See the introduction of this section for the way we use locations to express the semantics of programming languages. By *wait*(l) we denote that l is the location of a wait statement and *waitvalue*(l) denotes the specified waitvalue of that wait statement. The semantics of a wait statement is then specified by

$$\mathbf{J}\ at(l) \wedge wait(l) \rightarrow at(l)\ \mathbf{until}_{\delta(waitvalue(l))}\ after(l).$$

Remark 5.1. For the **J**-operator and the function δ transferring elements from a data domain to elements of the metric domain Δ, see the introduction of this section.

Remark 5.2. Being present at a location takes some time so the wait statement cannot be passed in 0 time units. In other words, even if the waitvalue is 0 the function δ will take care that this is mapped to a positive number to account for the time it takes to transfer control (cf. Appendix A in (Koymans, Shyamasundar, de Roever, Gerth and Arun-Kumar 1985) concerning this problem for the Ada delay statement).

Remark 5.3. If also an infinite waitvalue is allowed we add the following axiom for this special case:

$$at(l) \wedge wait\ (l) \wedge waitvalue(l) = \infty \rightarrow \mathbf{G}\ at(l).$$

5.4. Example 4: Terminal adaptor

This example is a mixture of message passing and real-time. It concerns a simplified terminal adaptor. On one side bytes are received from a data link operating on 512 bytes/second.

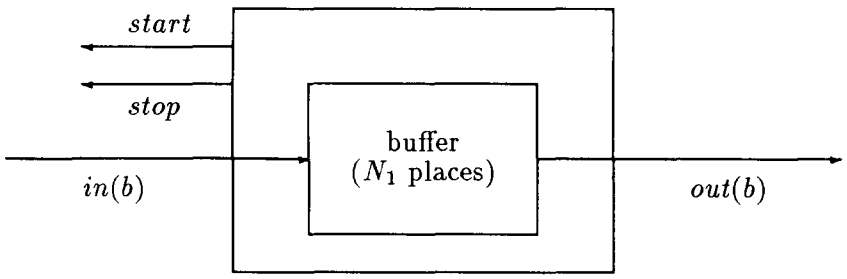


Figure 4. Terminal adaptor.

On the other side bytes are transmitted to a terminal with a rate of 300 bytes/second. The adaptor has a buffering capacity of N_1 bytes and it prevents buffer overflow through sending *stop* and *start* signals to the data link as soon as the buffer becomes more than 80% full, respectively more than 80% empty. It is assumed that after the sending of a *stop* signal at most N_2 bytes are sent by the data link (of course N_2 should be small compared to N_1). The data link may resume sending bytes only after it has received a *start* signal. Let *in*(*b*) denote the reception of byte *b* from the high-speed data link and *out*(*b*) the transmission of byte *b* to the terminal. The above is summarized in Figure 4.

The terminal adaptor is a perfect FIFO message passing system, so we suppose:

Unique Identification (MP1) for *in*,
No Creation and finite speed (MP2′) for *out* with respect to *in*,
perfectness (MP3′)
no simultaneous input and output (MP4a, b) for *in* and *out*,
FIFO ordering (MP5).

Additionally, the terminal adaptor obeys some real-time restrictions. First define

$$\mathit{buffered}(b) := \dot{\mathbf{P}}\ \mathit{in}(b) \wedge \neg\ \dot{\mathbf{P}}\ \mathit{out}(b)$$

to express that byte *b* is at the moment contained in the buffer of the terminal adaptor. We assume that transmission of bytes to the terminal is irregular (i.e., aperiodic), but within $\frac{1}{300}$ of a second:

$$\mathit{buffered}(b) \rightarrow \mathbf{F}_{<1/300}\ \exists\ b'\ \mathit{out}(b').$$

Because the buffer respects FIFO ordering this can be strengthened to

$$\mathit{buffered}(b) \wedge \neg\ \exists\ b'[\mathit{buffered}(b') \wedge \dot{\mathbf{P}}(\mathit{in}(b) \wedge \mathbf{P}\ \mathit{in}(b'))] \rightarrow \neg\ \exists\ b'\ \mathit{out}(b')\ \mathbf{until}_{<1/300}\ \mathit{out}(b)$$

where $\varphi\ \mathbf{until}_{<\delta}\ \psi$ is of course defined by

$$\exists \delta'[0 \ll \delta' \ll \delta \wedge \varphi\ \mathbf{until}_{\delta'}\ \psi].$$

The strengthened axiom above can be derived as an instance (taking $\varphi \equiv \exists b'\, out(b')$ and $\psi \equiv out(b)$) from

$$\neg\ \varphi\ \mathbf{until}\ \psi \wedge \mathbf{F}_{<\delta}\ \varphi \rightarrow \neg\ \varphi\ \mathbf{until}_{<\delta}\ \psi$$

(where $\neg\ \varphi\ \mathbf{until}\ \psi$ stems from the part about the FIFO ordering).

We now proceed with the other side, the reception of bytes from the data link. Define

$$stopped := (\neg\ start)\ \mathbf{since}\ stop$$

$$start\_\_stop\_\_interference := \Diamond_{<1/512}\ (stop \vee start)$$

(where $\Diamond_{<\delta}\ \varphi$ is defined by $\varphi \vee \mathbf{F}_{<\delta}\ \varphi$) to indicate that the reception was stopped (a *stop* signal was issued and since then no *start* signal has been issued), respectively a period (of length $1/512$) in which reception is interfered by issuing a *stop* or *start* signal. We can now specify the regular reception of bytes from the date link with period $1/512$, unless reception was stopped or interfered by a *stop* or *start* signal:

$$in(b) \wedge \neg\ stopped \wedge \neg\ start\_\_stop\_\_interference \rightarrow \neg\ \exists\ b'\ in(b')\ \mathbf{until}_{1/512}\ \exists\ b'\ in(b').$$

Remark 5.4. This axiom represents a conditional periodicity requirement. Therefore, the above axiom can also be written as

$$periodic(\exists\ b'\ in(b'),\ \frac{1}{512},\ \neg\ stopped \wedge \neg\ start\_\_stop\_\_interference).$$

(Recall from predicate logic that $\forall x[(P(x) \wedge Q) \rightarrow R]$ is equivalent with $(Q \wedge \exists x\, P(x)) \rightarrow R$ when Q and R do not contain x free.)

Remark 5.5. Note that $\neg\,(\neg\ start\ \mathbf{since}\ stop) \wedge \neg\ \Diamond_{<1/512}\ (stop \vee start)$ is equivalent with $\neg\ \mathbf{F}_{1/512}\ (\neg\ start\ \mathbf{since}\ stop) \wedge \neg\ \Diamond_{<1/512}\ start$ (the latter formulation was used in (Koymans, Kuiper and Zijlstra 1987)).

After a *stop* signal the data link need not immediately stop sending bytes (it can still send at most N_2 bytes). Nevertheless, the reception of bytes remains regular in such a period. To enforce this we also demand backward periodicity after the first byte after the last *start* signal:

$$in(b) \rightarrow \neg\ \exists\ b'\ in(b')\ \mathbf{since}\ start \vee \neg\ \exists\ b'\ in(b')\ \mathbf{since}_{1/512}\ \exists\ b'\ in(b').$$

After a *stop* signal at most N_2 bytes can be sent by the data link:

$$\neg\ start\ \mathbf{since}_{>N_2/512}\ stop \rightarrow \neg\ \exists\ b\ in(b)$$

where $\varphi\ \mathbf{since}_{>\delta}\ \psi$ is defined by

$$\exists \delta'[\delta \lessdot \delta' \wedge \varphi\ \mathbf{since}_{\delta'}\ \psi].$$

At last we should specify the generation of the *start* and *stop* signals. For convenience we assume that N_1 is divisible by 5. To indicate the situation that the buffer is at least 80% full, respectively at least 80% empty, we define

$$almostfull := \exists\, b_1 \ldots \exists\, b_{4/5N_1+1} \left[\bigwedge_{\substack{i,j=1 \\ i<j}}^{4/5N_1+1} b_i \neq b_j \wedge \bigwedge_{i=1}^{4/5N_1+1} buffered(b_i) \right]$$

$$almostempty := \neg\, \exists\, b_1 \ldots \exists\, b_{1/5N_1} \left[\bigwedge_{\substack{i,j=1 \\ i<j}}^{1/5N_1} b_i \neq b_j \wedge \bigwedge_{i=1}^{1/5N_1} buffered(b_i) \right].$$

Remark 5.6. N_1 is a fixed (constant) parameter in this specification so that the sequence of existential quantifiers in front of these formulas can be replaced by a sequence of fixed length.

Remark 5.7. When one allows the use of auxiliary data structures such as a queue, one simply could refer to the length of the queue representing the buffer. However, we consider the use of auxiliary data structures against the requirement of syntactical abstractness for specification languages (see Section 2). When one decides to use only logical and temporal operators combined with quantification over and equality in the data domain (in this case bytes), a bit more complex definitions like the ones above are unavoidable.

Now we should specify that the *start* and *stop* signals will be generated *as soon as* the buffer becomes (again) almost full, respectively almost empty. To express the as soon as aspect, we use the just-operator **J** (see the introduction of this section):

$$start \leftrightarrow \mathbf{J}\ almostempty$$

$$stop \leftrightarrow \mathbf{J}\ almostfull.$$

As one can see from these two axioms the *start* and *stop* signals are not essential and, using these two axioms, can be consequently replaced in the previous axioms by their equivalent right-hand sides. In other words, this specification can be given in a more abstract way only in terms of *in* and *out* without the implementation-oriented signals *start* and *stop*! This phenomenon occurs because we see systems as black boxes and hence only specify the outside, but on the other hand overviewing this outside from all sides (seeing the *whole* environment). In the case of the terminal adaptor, the *start* and *stop* signals are essential from an implementation viewpoint because the data link cannot see from its position how the other side (the terminal) is doing, in particular how fast the terminal adaptor transmits bytes at that side. Because the data link does not have this information, it is not able to stop in right time and start sending bytes again when necessary *by itself.*

5.5. Example 5: Mixing synchronous and asynchronous input

In this example we specify an object with two inputs and one output. The original informal specification is contained in (Denvir, Harwood, Jackson and Ray 1985):

The object has two inputs and one output. The output and one of the inputs respectively send and receive data in packets at regular intervals. The remaining input is asynchronous, that is, data appears at undetermined times.

The data packets which arrive at the synchronous input may be full or empty, and the object may only output data by forwarding packets from the synchronous input or filling an empty packet with data from the asynchronous input. All packets have the same size.

This is represented in Figure 5. The object, like the terminal adaptor of Example 4, has a mixture of message passing and real-time features. It seems the intention of the informal specification above that the periods of the output and the synchronous input are the same (in the picture represented by $\gamma > 0$). If the period of the output would be shorter than that of the synchronous input, the output will have to create packets at a certain moment and this violates the No Creation assumption for message passing systems. If, on the other hand, the period of the output would be longer than that of the synchronous input, the output cannot keep pace and packets will be lost eventually. As we interpret the above informal specification this seems not intended because that specification suggests that the object functions as a *perfect* message passing system. Furthermore, we assume finite speed for the passing of packets. Because of the synchrony of the output and one of the inputs this leads to a fixed delay $\delta > 0$. This delay δ represents a kind of processing time to pass or possibly fill a packet. The message passing aspect of the object is somewhat unusual because only one output is coupled to two inputs. The most important input is, however, the synchronous one and the asynchronous one only functions in exceptional cases (an empty packet on the synchronous input). Therefore, the following message passing properties hold between the two inputs and the output: No Creation and finite speed hold between the output and both inputs, FIFO holds for the output and the synchronous input while perfectness only holds for nonempty packets on the synchronous input. These message passing properties will be a consequence of stronger real-time properties given below. We do assume no simultaneous input and output:

$$in_s(p) \wedge in_s(p') \rightarrow p' = p$$

$$in_a(p) \wedge in_a(p') \rightarrow p' = p$$

$$out(p) \wedge out(p') \rightarrow p' = p.$$

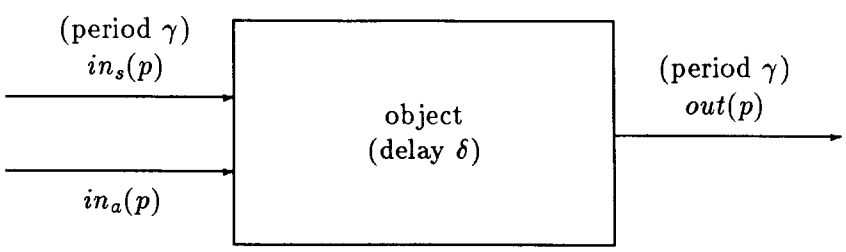


Figure 5. Mixing synchronous and asynchronous input.

Also unique identification is supposed. Because the inputs are not clearly separated, but are mixed in this case, we must not only assume unicity for both inputs separately but also for the inputs between each other:

$$in_s(p) \vee in_a(p) \rightarrow \neg\, \mathbf{P}\, (in_s(p) \vee in_a(p))$$
$$\neg\, (in_s(p) \wedge in_a(p)).$$

Recall from the introduction of this section that the No Creation assumption on message passing systems consisted of two parts: no new messages and no duplicates. The no new messages part will follow from the real-time requirements below, but the no duplicates part is independent from the message passing relation between the output and the two inputs described above. So, we demand for the output:

$$out(p) \rightarrow \neg\, \mathbf{P}\, out(p).$$

We can now turn to the real-time requirements of the object. Using the abbreviation *periodic*(e, δ) defined in the introduction of this section, regularity of the output, respectively synchronous input, is required by

$$periodic(\exists\, p'\ in_s(p'),\ \gamma)$$

and

$$periodic(\exists\, p'\ out(p'),\ \gamma).$$

The following two real-time requirements concern perfectness with a delay of δ differentiating the cases of a nonempty and empty packet on the synchronous input:

$$in_s(p) \wedge \neg\, empty(p) \rightarrow \mathbf{F}_\delta\ out(p)$$
$$in_s(p) \wedge empty(p) \rightarrow \mathbf{F}_\delta\ (out(p) \vee \exists\, p'\ [out(p') \wedge \mathbf{P}\ in_a(p')]).$$

Remark 5.8. The latter axiom allows that a packet arrives on the asynchronous input at the very last moment. This is not quite in accordance with the idea that the delay δ represents a kind of processing time to pass or possibly fill a packet. More tailored towards this idea would be the axiom

$$in_s(p) \wedge empty(p) \rightarrow \mathbf{F}_\delta\ out(p) \vee \exists\, p'\ [\dot{\mathbf{P}}\ in_a(p') \wedge \mathbf{F}_\delta\ out(p')],$$

that is, getting the **P**-operator out of the scope of the $\mathbf{F}_\delta$.

Remark 5.9. Both axioms together (with the obvious change in case the alteration suggested in the previous remark is taken into account) guarantee that

$$in_s(p) \rightarrow \mathbf{F}_\delta(out(p) \vee \exists\, p'\ [out(p') \wedge \mathbf{P}\ in_a(p')]),$$

so in particular

$$in_s(p) \rightarrow \mathbf{F}_\delta \exists p'\ out(p').$$

Remark 5.10. Because of Remark 5.9 and regularity of the output with period γ, the axiom for regularity of the synchronous input with period γ can be weakened to

$$in_s(p) \rightarrow \mathbf{F}_\gamma \exists p'\ in_s(p').$$

The reason is that $in_s(p) \wedge \mathbf{F}_{<\gamma} \exists p'\ in_s(p')$ implies by Remark 5.9

$$\mathbf{F}_\delta(\exists p'\ out(p') \wedge \mathbf{F}_{<\gamma} \exists p'\ out(p'))$$

which contradicts the regularity of the output with period γ.

Our last axiom ensures that output does not start too early, to be precise only after a delay δ after the first packet on the synchronous input:

$$\neg \exists p\ \dot{\mathbf{P}}\ in_s(p) \rightarrow \mathbf{F}_\delta \neg \exists p'\ \dot{\mathbf{P}}\ out(p').$$

An equivalent formulation of this axiom looks backwards:

$$\exists p'\ \dot{\mathbf{P}}\ out(p') \rightarrow \mathbf{P}_\delta \exists p\ \dot{\mathbf{P}}\ in_s(p).$$

Now we can show that the remaining message passing properties are implied by the above real-time requirements. First, an obvious strengthening of Remark 5.9 gives

$$in_s(p) \rightarrow \mathbf{F}_\delta((out(p) \wedge \mathbf{P}_\delta\ in_s(p)) \vee \exists p'[out(p') \wedge \mathbf{P}\ in_a(p')]).$$

So, each packet on the synchronous input leads after a delay δ to the output of either that packet or an earlier packet from the asynchronous input. Since the synchronous input and the output have the same period γ these packets caused by the synchronous input make up for all packets on the output from a delay δ after the first packet on the synchronous input (there can be no packets in between since the output is regular and there can be no packets simultaneously with those generated by the synchronous input because no simultaneous output is assumed). The last axiom ensures that before a delay δ after the first packet on the synchronous input there can be no packet on the output. Thus, the only packets on the output are those generated by a packet on the synchronous input as formulated by the above formula. Inspecting this formula we immediately can conclude no creation of new packets and finite speed since either $out(p) \wedge \mathbf{P}_\delta\ in_s(p)$ or $out(p') \wedge \mathbf{P}\ in_a(p')$ holds. In fact, we showed that $out(p) \rightarrow \mathbf{P}_\delta\ in_s(p) \vee \mathbf{P}\ in_a(p)$. No duplication of packets was already formulated separately and takes care that a packet from the asynchronous input cannot be taken twice to fill an empty packet from the synchronous input. FIFO ordering for packets from the synchronous input follows because the above implies that a packet from the synchronous input will be output after a delay δ or not at all as formulated by the formula

$$in_s(p) \rightarrow \mathbf{F}_\delta\ out(p) \vee \mathbf{G} \neg\ out(p).$$

Perfectness for nonempty packets of the synchronous input follows already solely from the axiom about nonempty packets at the synchronous input.

5.6. *Example 6: Abstract transmission medium*

A transmission medium can be considered as a message passing system where the input and output are called transmit, respectively arrive and the data consists of signals. We assume the following aspects of message passing systems: unique identification of signals, no creation of signals and finite transmission speed, basic liveness, no simultaneous input and output. As given in the introduction of this section these can be formulated respectively by:

$$transmit(s) \rightarrow \neg\ \mathbf{P}\ transmit(s)$$

$$arrive(s) \rightarrow \mathbf{P}\ transmit(s) \wedge \neg\ \mathbf{P}\ arrive(s)$$

$$\mathbf{G}\ \mathbf{F}\ \exists\ s\ transmit(s) \rightarrow \mathbf{F}\ \exists\ s\ arrive(s)$$

$$transmit(s) \wedge transmit(s') \rightarrow s' = s$$

$$arrive(s) \wedge arrive(s') \rightarrow s' = s.$$

The characteristic feature of the transmission medium on top of being a particular kind of message passing system is the requirement that it is not too lazy, that is, there exists a fixed period γ in which the transmission medium attempts to transmit at least one signal (successfully or not). So, when there are no other signals to be transmitted, γ represents the maximum time for which the attempt to transmit a signal can be delayed. Such a requirement is needed to enable higher-level protocols to time-out on signals sent but not yet received and start retransmission. This is formulated by

$$\exists\ \gamma\ \mathbf{G}\ (\exists\ s\ [\dot{\mathbf{P}}\ transmit(s) \wedge \neg\ \dot{\mathbf{P}}\ arrive(s)] \rightarrow \exists\ s'\ [\dot{\mathbf{P}}\ transmit(s') \wedge \neg\ \dot{\mathbf{P}}\ arrive(s') \wedge \mathbf{G}_{>\gamma} \neg\ arrive(s')])$$

where $\mathbf{G}_{>\delta}\ \varphi$ is defined by

$$\forall \delta' [\delta \lll \delta' \rightarrow \mathbf{G}_{\delta'}\ \varphi].$$

In the axiom above s' represents one signal which has been attempted to transmit in a particular period γ. If this transmission was successful, $\mathbf{F}_{\leq\gamma}\ arrive(s')$ holds (where $\mathbf{F}_{\leq\delta}\ \varphi$ is defined by $\mathbf{F}_\delta\ \varphi \vee \mathbf{F}_{<\delta}\ \varphi$), otherwise $\mathbf{A} \neg\ arrive(s')$ (see Theorem 4.1 in Section 4 for the definition of $\mathbf{A}$) holds. To prove this we note the following. Since $\mathbf{A} \neg\ \varphi$ is equivalent over linear orders with $\neg\ \dot{\mathbf{P}}\ \varphi \wedge \mathbf{G} \neg\ \varphi$ and $\neg\ \dot{\mathbf{P}}\ arrive(s')$ is given, it is sufficient to prove

$$\mathbf{G}_{>\gamma} \neg\ arrive(s') \rightarrow \mathbf{F}_{\leq\gamma}\ arrive(s') \vee \mathbf{G} \neg\ arrive(s').$$

Now, this is an instance of $\mathbf{G}_{>\gamma} \neg \psi \rightarrow \mathbf{F}_{\leq\gamma} \psi \vee \mathbf{G} \neg \psi$ which is a theorem of M as is shown by the following derivation:

1. $\exists\delta[0 \ll \delta \underline{\ll} g \wedge \mathbf{F}_\delta \psi] \leftrightarrow \mathbf{F}_{\leq\gamma} \psi$	(definition $\mathbf{F}_{\leq\gamma} \psi$)
2. $\neg\forall\delta[0 \ll \delta \underline{\ll} \gamma \rightarrow \mathbf{G}_\delta \neg \psi] \leftrightarrow \mathbf{F}_{\leq\gamma} \psi$	(1, predicate logic, $M0$)
3. $\forall\delta[\gamma \ll \delta \rightarrow \mathbf{G}_\delta \neg \psi] \wedge \forall\delta[0 \ll \delta \underline{\ll} \gamma \rightarrow \mathbf{G}_\delta \neg \psi] \rightarrow \forall\delta[0 \ll \delta \rightarrow \mathbf{G}_\delta \neg \psi]$	(predicate logic)
4. $\mathbf{G}_0 \neg \psi \leftrightarrow \top$	($M5$a)
5. $\forall\delta[0 \ll \delta \rightarrow \mathbf{G}_\delta \neg \psi] \leftrightarrow \forall\delta\, \mathbf{G}_\delta \neg \psi$	(4, predicate logic)
6. $\forall\delta[0 \ll \delta \rightarrow \mathbf{G}_\delta \neg \psi] \leftrightarrow \mathbf{G} \neg \psi$	(5, $M0$)
7. $\forall\delta[\gamma \ll \delta \rightarrow \mathbf{G}_\delta \neg \psi] \wedge \forall\delta[0 \ll \delta \underline{\ll} \gamma \rightarrow \mathbf{G}_\delta \neg \psi] \rightarrow \mathbf{G} \neg \psi$	(3, 6)
8. $\mathbf{G}_{>\gamma} \neg \psi \rightarrow \neg \forall\delta[0 \ll \delta \underline{\ll} \gamma \rightarrow \mathbf{G}_\delta \neg \psi] \vee \mathbf{G} \neg \psi$	(7, definition $\mathbf{G}_{>\gamma} \neg \psi$)
9. $\mathbf{G}_{>\gamma} \neg \psi \rightarrow \mathbf{F}_{\leq\gamma} \psi \vee \mathbf{G} \neg \psi$	(2, 8)

In this example we needed quantification over the metric domain.

5.7. Example 7: Real-time communication constructs

In this example we describe asynchronous message passing by means of the send and receive constructs. Our specific form of the send and receive constructs is inspired by CHILL (see (Chill 1980)). The send construct has an associated signal which represents the data to be sent. Each signal has a unique destination and every signal sent will eventually reach its destination. The receive construct consists of a selection of signals that it may accept. The selection is between signals that have been sent to the process to which this receive construct belongs (that must be their destination), that have arrived and that have not been selected before. After a choice has been made, control transfers to the corresponding part of the receive construct. So, for a receive construct we can differentiate two phases:

1. wait (possibly forever) for a signal that can be accepted (one of the listed selection possibilities),
2. choose one of the acceptable signals and take the branch of that accepted signal.

In case of a timed receive construct the possibility of a time-out is added that restricts the time the receiving process is going to wait for a signal matching one of its selection possibilities to arrive. For real-time applications the communication constructs of (asynchronous) send and timed receive are the most useful choices because they do not lead to deadlock possibilities (the sender continues and the receiver times out). The send and receive constructs are high-level communication primitives and are usually implemented on a network providing reliable communication by using time-out and retransmission for unreliable transmission media like those of Example 6. Notice that send and receive resemble *in*, respectively *out*, of a perfect message passing system. The main difference, however, is that the receiver explicitly accepts signals at times *chosen by itself*. In other words: the possibility

to output a message is under control of the environment instead of the system. We start by specifying the effect of a send statement:

$$at(l) \wedge send(l) \rightarrow at(l) \textbf{ until } after(l).$$

We use similar conventions about locations as we used in Example 3. $send(l)$ indicates that the location l contains a send statement. This axiom simply states that the send statement takes some finite time, and this is exactly the essence of an asynchronous send: the sender just continues in contrast with synchronous communication such as a rendezvous in Ada (see (Programming Language Ada 1983)). The signal that is the result of the send statement at location l will be represented by the function $signal(l)$. An alternative for this would be to put this explicitly in the predicate *send*, but in that case it should be additionally stated that only *one* signal is generated for each send statement:

$$send(l, s) \wedge send(l, s') \rightarrow s' = s.$$

We prefer the use of the predicate $send(l)$ and the function $signal(l)$ because then it is implicit that a send statement can generate only one signal. The fact that a signal s is sent can be expressed by

$$sent(s) := \exists\, l\, [\mathbf{J}\ after(l) \wedge send(l) \wedge signal(l) = s].$$

Here we use the just-operator to indicate that the moment of sending coincides precisely with the moment that the send statement has just been passed. Because send statements can be executed simultaneously at different places (locations in different processes) in the program, and similarly for receive statements, we cannot suppose the no simultaneous input assumption. We want the data passed to be unique, so we must demand that simultaneously executed send statements generate different signals:

$$send(l) \wedge send(l') \wedge signal(l) = signal(l') \rightarrow l' = l.$$

We now turn to the receiving side. As we indicated above, the message passing relation between the sender and the receiver is somewhat nonstandard because the receiver chooses the time to make a selection between acceptable signals. This selection process is also a special one: Only certain signals can be accepted. This is expressed by the predicate $selectable(s, l)$. There are several choices for the definition of this predicate depending on the intended possibilities to select signals, but the signal s should at least conform (either syntactically or semantically) to one of the possible choices of that particular receive statement (i.e., the one at location l) and the destination of s should be the process in which this receive statement (i.e., the location l) occurs. With a receive statement at location l and a signal s we associate the special location $choice(s, l)$ representing the location where control is transferred to when signal s is chosen to be accepted at l. For these special locations $choice(s, l)$ we again impose a uniqueness assumption:

$$choice(s, l) = choice(s', l') \rightarrow s' = s \wedge l' = l.$$

A signal s can be chosen to be accepted at l if it is selectable, has been sent and was not chosen before. So define

$$choosable(s, l) := selectable(s, l) \wedge \mathbf{P}\ sent(s) \wedge \neg\ \mathbf{P}\ \exists\ l'\ at(choice(s, l')).$$

The fact that we can model that a signal s has been chosen before by $\mathbf{P}\ \exists l'\ at(choice(s, l'))$ depends crucially on the uniqueness assumption for the locations $choice(s, l)$. To see this, consider the following program with three processes:

P_1:: *SEND* 0 *TO* P_3
P_2:: *SEND* 0 *TO* P_3
P_3:: *RECEIVE*
0: ...
1: ... ;
RECEIVE
0: ...
1: ...

Let s_1 and s_2 be the signals sent from P_1 and P_2 respectively, and l one of the two receive statements in P_3, then

$$choice(s_i, l) = choice(s_{3-i}, l) \qquad \text{for } 1 \leq i \leq 2.$$

So, if s_1 is accepted in P_3 first, $\mathbf{P}\ \exists l'\ at(choice(s_2, l'))$ will hold although s_2 has not been chosen yet.

To arrive at a location $choice(s, l)$, s must have been choosable at l:

$$\mathbf{J}\ at(choice(s, l)) \rightarrow choosable(s, l).$$

The (nontimed) receive statement can now be described by the following two axioms:

$$at(l) \wedge receive(l) \rightarrow at(l)\ \textbf{unless}\ \exists\ s\ at(choice(s, l))$$

$$at(l) \wedge receive(l) \wedge \exists\ s\ choosable(s, l) \rightarrow \mathbf{F}\ \exists\ s'\ at(choice(s', l)).$$

In the case of a timed receive statement there is the additional possibility to transfer control to the special else-location after timervalue (cf. the waitvalue of a wait statement in Example 3) time units have elapsed. Combined with the two axioms above for the nontimed case this leads to the axiom

$$\mathbf{J}\ at(l) \wedge timedreceive(l) \rightarrow$$
$$at(l)\ \textbf{until}_{\delta(timervalue(l))}\ at(else(l)) \vee$$
$$at(l)\ \textbf{until}_{<\delta(timervalue(l))}$$
$$((at(l) \wedge \exists\ s\ choosable(s, l))\ \textbf{until}\ \exists\ s'\ at(choice(s', l))).$$

Note that the choice to take the else-branch is always possible because it is not observable whether a signal has arrived at its destination or not. In other words, we know nothing about the speed of the reliable communication network. It would be realistic to impose an upper bound on the time for signals to arrive (the maximum transmission time). In that case the else-branch can only be taken if we add that there could not have arrived a signal within timervalue time units. This can be done by adding the following conjunct to the first clause of the disjunction in the axiom above (*maxtt* represents the maximum transmission time):

$$\wedge\ \neg\ \exists\ s[selectable(s,\ l)\ \wedge\ \neg\ \mathbf{P}\ \exists\ l'\ at(choice(s,\ l'))\ \wedge\ \mathbf{F}_{\delta(timervalue(l))}\ \mathbf{P}_{>maxtt}\ sent(s)]$$

where $\mathbf{P}_{>\delta}\ \varphi$ is defined by

$$\exists\delta'\ [\delta \lll \delta'\ \wedge\ \mathbf{P}_{\delta'}\ \varphi].$$

In the same way one can introduce a minimum transmission time by incorporating such a *mintt* in the definition of *choosable*(*s*, *l*):

$$selectable(s,\ l)\ \wedge\ \mathbf{P}_{>mintt}\ sent(s)\ \wedge\ \neg\ \mathbf{P}\ \exists\ l'\ at(choice(s,\ l')).$$

A (timed) receive statement can choose between several signals to accept. A fairness assumption can be added for these choices, relating to the locations *choice*(*s*, *l*).

5.8. Example 8: Continuously changing state variables

In the previous examples we concentrated on events since these are very important for real-time systems. In case state variables also play an important role, for example, in case of process control systems, it is still often the case that not the variable itself is the dominant feature but a certain event or condition involving this state variable. A typical example is a continuous physical variable like temperature. (Note: Continuous here does not refer so much to the possibility of allowing a continuous range of values for temperature, but to the fact that temperature may change continuously in time.) Usually we are not interested in the absolute value of this state variable as such but more in the fact whether it stays within certain bounds, for example, the system should only react when the condition $temperature \leq maxtemp$ becomes false. An instance where such a condition is crucial is a process where this condition leads to irreversible phenomena (such as a chemical chain reaction) that occur immediately. Whenever such a catastrophe occurs, emergency measures (such as a *shutdown*) are required. Suppose that *reactime* is the required reaction time and that *shutdown* is the required reaction, then such a requirement can be specified in metric temporal logic by

$$\mathbf{J}(temperature > maxtemp) \rightarrow \mathbf{F}_{<reactime}\ shutdown.$$

Here we use the just-operator to catch the exact moment when the condition *temperature* $\leq$ *maxtemp* changes from true to false.

One could argue that this behavior can be captured by a discrete system that takes samples of temperature and that uses only two events, *catastrophic* and *critical*, that have the following correspondence:

$$catastrophic \equiv temperature > maxtemp$$

$$critical \equiv maxtemp - temperature < \epsilon$$

where ϵ is a constant depending on the rate of change of temperature and the chosen sample time. The point is that even if the possible rate of change is exactly known beforehand and the chosen sample time is sufficiently fast, it cannot be determined in the case that temperature has a value in between *maxtemp* $- \epsilon$ and *maxtemp* whether event *catastrophic* will occur. For that reason, if one wants to stay at the safe side, this entails that a shutdown should be performed whenever the event *critical* (instead of *catastrophic*) occurs. In other words, the threshold has been lowered from the occurrence of the event *catastrophic*, as originally intended, to the event *critical*. In temporal logic the above can be rephrased by the validity of

$$catastrophic \rightarrow \mathbf{P}\ critical$$

but not

$$critical \rightarrow \mathbf{F}\ catastrophic.$$

So, the conclusion must be that discrete systems can only model continuously changing state variables to a certain extent. Therefore, if one wants to specify the behavior of such state variables completely, a temporal logic that can reason about continuous time domains is essential. Metric temporal logic caters for this possibility.

6. Conclusions

We extended temporal logic with metric operators derived from their qualitative versions described in Section 3. We showed how these metric operators could be usefully applied to the formal specification of real-time systems. (Burgess 1984) Section 6 contains an alternative proposal for metric temporal logic where time is structured as an ordered Abelian group. From a philosophical viewpoint, the idea that duration of time is expressed as an element of the time domain itself seems unnatural. Also technically, the natural addition on a time domain may not be sufficient for determining the distance between any two points, as is exemplified by the points (0, 1) and (1, 0) in Example 4.1 of Section 4. When only interested in qualitative aspects of distances, however, Tarski's qualitative geometry (Tarski 1969) suggests models $(T, <, E)$ where E is the equidistance-predicate ($Exyuv$ if and only if x and y have the same distance as u and v). An interesting question connecting this

approach with metric temporal logic is: How should $<$ and E be axiomatized to describe models $(T, <, E)$ that allow a *representation* in terms of our metric point structures such that $Exyuv \Leftrightarrow d(x, y) = d(u, v)$? Another alternative for expressing quantitative timing properties is dynamic logic (see (Harel 1984)) with one atomic program 'successor' S. But, already for the expression of bounded response time we need an *infinitary* dynamic logic (Goldblatt 1982):

$$\bigvee_n [S^*] \left(p \rightarrow \bigvee_{i<n} < S^i > q \right).$$

This approach is only suitable for *discrete* structures, but our philosophy behind metric temporal logic required that the qualitative fragment concerning *all* point structures should be nicely embedded. This makes sense in practice too, because real-time systems may contain nondiscrete elements such as analog devices for handling continuous physical entities like temperature (see Section 2 and example 8 of Section 5).

A review of formal methods for describing real-time systems is given in (Joseph and Goswami 1988). As far as we know, (Bernstein and Harter 1981) was the first paper (using temporal logic) to specify timing characteristics of real-time systems formally. Their approach differs at several points from ours. Firstly, they use only real-time operators related to temporal implication instead of the more powerful operators of metric temporal logic. Secondly, they use the interleaving model. Consequently their method is restricted to uniprocessor implementations. Thirdly, their method is limited to specific safety properties. (Pnueli and Harel 1988) contains a brief account of some attempts to use temporal logic for the specification of real-time systems. The computational model used is a timed interleaving model where enabled transitions have associated lower and upper bounds within which they must be taken. It considers two possible extensions of temporal logic to deal with real-time. The first adds a global clock as an explicit variable to which the specification may refer. The second approach introduces quantitative temporal operators and is very much akin to metric temporal logic. For specifying synchronous systems it recommends the use of a discrete time domain (such as the natural numbers) and for asynchronous systems a dense time domain (such as the rationals). One of the methods using the first approach is that of Ostroff (1987, 1989). It introduces a distinguished variable t representing the clock. A typical formula of his logic RTTL (Real-Time Temporal Logic) is the following:

$$\varphi \wedge t = T \rightarrow \Diamond (\psi \wedge t \leq T + 5)$$

where T is a global variable.

The semantics of this formula corresponds to that of the metric temporal logic formula

$$\varphi \rightarrow \Diamond_{\leq 5} \psi.$$

As is obvious from this example, metric temporal logic provides a more concise and natural way of specifying real-time properties: the explicit clock variable is against the original philosophy of temporal logic to abstract from time as much as possible (and in the case of real-time it is sufficient to add only terms for expressing time units as in the metric

temporal logic formula above). On the other hand, RTTL is based on the work of Manna and Pnueli so that a sound proof system based on their work is immediately available. An example using the second approach is (Ghezzi, Mandrioli and Morzenti 1989) incorporating an executable specification language. Another formal approach to the specification of real-time systems, not based on temporal logic, is the Real-Time Logic (RTL) of Jahanian and Mok (1986, 1987). Events are central in RTL and reasoning about real-time systems is based on assertions about the occurrences of events which are mapped by the occurrence function into the time domain of the natural numbers. The use of RTL is restricted to the specification of safety properties.

The examples in Section 5 showed how several aspects of real-time systems can be specified with metric temporal logic, ranging from very simple real-time constructs and systems to combined message passing/real-time systems and semantics for real-time communication constructs of concurrent programming languages. The resulting specifications are elegant and rather directly formalize our intuition about the timing aspects of real-time systems.

Regarding future developments, it remains to be seen how we can apply metric temporal logic to medium and large scale examples. Before this can be done it must be sorted out how we can embed such a specification formalism into a method that supports hierarchical development and caters for the description of complex data structures.

Acknowledgments

The author wishes to thank Loe Feijs, Willem-Paul de Roever and Job Zwiers for their comments and suggestions on the first draft of this paper. The remarks and suggested improvements of an anonymous referee were also very helpful.

References

The Programming Language Ada, Reference Manual. 1983. *Lecture Notes in Computer Science*, vol. 155, Berlin: Springer.

Barringer, H. and Kuiper, R. 1985a. Hierarchical Development of Concurrent Systems in a Temporal Logic Framework. Proceedings of a Seminar on Concurrency, Carnegie Mellon University, Pittsburgh, pp. 35–61, July 1984; *Lecture Notes in Computer Science*, vol. 197, Berlin: Springer.

Barringer, H. and Kuiper, R. 1985b. Towards the Hierarchical Temporal Logic Specification of Concurrent Systems. In Denvir, T. et al., *Lecture Notes in Computer Science*, vol. 207, Berlin: Springer, pp. 156–183.

Barringer, H., Kuiper, R. and Pnueli, A. 1984. Now You May Compose Temporal Logic Specifications. *Proceedings of the Sixteenth ACM Symposium on the Theory of Computing*, pp. 51–63.

Barringer, H., Kuiper, R. and Pnueli, A. 1986. A Really Abstract Concurrent Model and its Temporal Logic. *Proceedings of the Thirteenth ACM Symposium on the Principles of Programming Languages*, pp. 173–183.

Benthem, J.F.A.K. van. 1983. *The Logic of Time*. Dordrecht: Reidel.

Bernstein, A. and Harter, P.K. Jr. 1981. Proving Real-Time Properties of Programs with Temporal Logic, *Proceedings of the Eighth ACM Symposium on Operating System Principles*, pp. 1–11.

Burgess, J.P. 1984. Basic Tense Logic. In *Handbook of Philosophical Logic*, vol. II, (Gabbay and Guenthner, eds.), pp. 89–133.

CHILL Recommendation Z.200 (CHILL Language Definition). 1980. C.C.I.T.T. Study Group XI.

Cocchiarella, N.B. 1984. Philosophical Perspectives on Quantification in Tense and Modal Logic. In *Handbook of Philosophical Logic*, vol. II, (Gabbay and Guenthner, eds.), pp. 309–353.

Denvir, T., Harwood, W., Jackson, M. and Ray, M. 1985. The Analysis of Concurrent Systems. Proceedings of a Tutorial and Workshop, Cambridge University, September 1983, *Lecture Notes in Computer Science*, vol. 207, Berlin: Springer.

Gabbay, D. and Guenthner, F., (eds.). 1984. *Handbook of Philosophical Logic*, vol. II. Dordrecht: Reidel.

Garson, J.W. 1984. Quantification in Modal Logic. In *Handbook of Philosophical Logic*, vol. II, (Gabbay and Guenthner, eds.) pp. 249–307.

Ghezzi, C., Mandrioli, D. and Morzenti, A. 1989. TRIO: A Logic Language for Executable Specifications of Real-Time Systems. Report 89-006, Dipartimento di Elettronica, Politecnico di Milano.

Goldblatt, R. 1982. Axiomatising the Logic of Computer Programming. *Lecture Notes in Computer Science*, vol. 130, Berlin: Springer.

Harel, D. 1984. Dynamic Logic, In *Handbook of Philosophical Logic*, vol. II, (Gabbay and Guenthner, eds.), pp. 497–604.

Hooman, J. and Widom, J. 1989. A Temporal-Logic Based Compositional Proof System for Real-Time Message Passing, to appear in Proceedings of the Conference on Parallel Architectures and Languages Europe (PARLE) '89, *Lecture Notes in Computer Science*, vol. 366 Berlin: Springer, pp. 424–441.

Jahanian, F. and Mok, A.K. 1986. Safety Analysis of Timing Properties in Real-Time Systems. *IEEE Transactions on Software Engineering*, 12, pp. 890–904.

Jahanian, F. and Mok, A.K. 1987. A Graph-Theoretic Approach for Timing Analysis and its Implementation. *IEEE Transactions on Computers*, vol. C-36, pp. 961–975.

Joseph, M. and Goswami, A. 1988. Formal Description of Real-Time Systems: A Review. Research Report RR129, Department of Computer Science, University of Warwick.

Katz, S. and Peled, D. 1987. Interleaving Set Temporal Logic, *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing*, pp. 178–190.

Koymans, R., Kuiper, R. and Zijlstra, E. 1987. Specifying Message Passing and Real-Time Systems with Real-Time Temporal Logic. *Proceedings of the Fourth Annual ESPRIT Conference*, Amsterdam: North-Holland, pp. 311–324.

Koymans, R. 1989. Specifying Message Pasing and Time-Critical Systems with Temporal Logic. Ph.D. Thesis, Eindhoven University of Technology.

Koymans, R., Shyamasundar, R.K., de Roever, W.-P., Gerth, R. and Arun-Kumar, S. 1985. Compositional Semantics for Real-Time Distributed Computing. Proceedings of the Workshop on Logics of Programs '85, *Lecture Notes in Computer Science*, vol. 193, Berlin: Springer, pp. 167–189, (extended version appeared in Information and Computation, Volume 79, Number 3, pp. 210–256, Academic Press, December 1988).

Koymans, R., Vytopil, J. and de Roever, W.-P. 1983. Real-Time Programming and Asynchronous Message Passing. *Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing*, pp. 187–197.

Lamport, L. 1983a. What Good is Temporal Logic? *Proceedings of Information Processing (IFIP) '83*, (Mason, R., ed.), Amsterdam: North-Holland, pp. 657–668.

Lamport, L. 1983b. Specifying Concurrent Program Modules, *ACM Transactions on Programming Languages and Systems* (TOPLAS), vol. 5, pp. 190–223.

Lichtenstein, O., Pnueli, A. and Zuck, L. 1985. The Glory of The Past. Proceedings of the Workshop on Logics of Programs '85, *Lecture Notes in Computer Science*, vol. 193, Berlin: Springer, pp. 196–218.

Manna, Z. and Pnueli, A. 1987. A Hierarchy of Temporal Properties. Deparment of Computer Science, Stanford University, Report No. STAN-CS-87-1186.

Moszkowski, B. and Manna, Z. 1984. Reasoning in Interval Temporal Logic, Proceedings of AMC/NSF/ONR Workshop on Logics of Programs, *Lecture Notes in Computer Science*, vol. 164, Berlin: Springer, pp. 371–383.

Moszkowski, B. 1983. Reasoning about Digital Circuits. Ph.D. Thesis, Department of Computer Science, Stanford University.

Moszkowski, B. 1986. *Executing Temporal Logic Programs*. Cambridge: Cambridge University Press.

Ostroff, J.S. 1987. Real-Time Computer Control of Discrete Event Systems Modelled by Extended State Machines: A Temporal Logic Approach. Ph.D. Thesis, Department of Electrical Engineering, University of Toronto.

Ostroff, J.S. 1989. *Temporal Logic for Real-Time Systems*. Advanced Software Development Series, Research Studies Press Limited (marketed by John Wiley and Sons), England.

Penczek, W. 1988. A Temporal Logic for Event Structures, *Fundamenta Informaticae*, XI, pp. 297–326.

Prior, A. 1967. *Past, Present and Future*. London: Oxford University Press.

Pnueli, A. and Harel, E. 1988. Applications of Temporal Logic to the Specification of Real-Time Systems, Proceedings of a Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems, (Joseph, M., ed.), University of Warwick, *Lecture Notes in Computer Science*, vol. 331, Berlin: Springer, pp. 84–98.

Pnueli, A. 1977. The Temporal Logic of Programs, *Proceedings of the Eighteenth Symposium on the Foundation of Computer Science*, pp. 46–57.

Pinter, S. and Wolper, P. 1984. A Temporal Logic for Reasoning about Partially Ordered Computations. *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing*, pp. 28–37.

Schwartz, R.L., Melliar-Smith, P.M. and Vogt, F.H. 1983. An Interval Logic for Higher-Level Temporal Reasoning. *Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing*, pp. 173–186.

Stirling, C. 1987. Comparing Linear and Branching Time Temporal Logics. ECS-LFCS-87-24, Laboratory for Foundations of Computer Science, Department of Computer Science, University of Edinburgh.

Tarski, A. 1969. What is Elementary Geometry? In Hintikka, J. (ed.), *The Philosophy of Mathematics*, London: Oxford University Press, pp. 164–175.

Thomas, W. 1986. Safety- and Liveness-Properties in Propositional Temporal Logic: Characterizations and Decidability. Schriften zur Informatik und Angewandten Mathematik, Bericht Nr. 116, Rheinisch-Westfälische Technische Hochschule Aachen.

Wolper, P. 1986. Expressing Interesting Properties of Programs in Propositional Temporal Logic. *Proceedings of the Thirteenth ACM Symposium on the Principles of Programming Languages*, pp. 184–193.