Temporal Logics for Real-Time System Specification

P. Bellini, R. Mattolini, P. Nesi

Department of Systems and Informatics, University of Florence

Via S. Marta 3, 50139 Firenze, Italy, tel.: +39-055-4796523, fax.:+39-055-4796363

email: nesi@ingfi1.ing.unifi.it, www: http://www.dsi.unifi.it/~nesi

The specification of reactive and real-time systems must be supported by formal, mathematically-founded methods to be satisfactory and reliable. Temporal logics have been used to this end for several years. Temporal logics allow the specification of system behavior in terms of logical formulae – including temporal constraints, events, and the relationships between the two. In the last 10 years, temporal logics have reached a high degree of expressiveness. Most of the temporal logics proposed in the last few years can be used for specifying of reactive systems, although not all are suitable for specifying real-time systems. In this paper, we present a series of criteria for assessing the capabilities of temporal logics for the specification, validation, and verification of real-time systems. Among the criteria are the logic's expressiveness, the logic's order, the presence of a metric for time, the type of temporal operators, the fundamental time entity, and the structure of time. We examine a selection of temporal logics proposed in the literature. To make the comparison clearer a set of typical specifications has been identified and used with most of the temporal logics considered, thus presenting the reader with a number of real examples.

Categories and Subject Descriptors: F.4.1 [**MATHEMATICAL LOGIC AND FORMAL LANGUAGES**]: Mathematical Logic—*Temporal logic, Modal logic*; D.2.1 [**SOFTWARE ENGINEERING**]: Requirements/Specifications—*Languages*; D.2.4 [**SOFTWARE ENGINEERING**]: Software/Program Verification—*Formal methods*; J.7 [**COMPUTERS IN OTHER SYSTEMS**]: Real time

General Terms: logic specification languages

Additional Key Words and Phrases: temporal logics, real-time, modal logic, metric of time, specification model, reactive systems, temporal constraints, temporal relationships

1. INTRODUCTION

In the last few years, several techniques, tools, and models for the formal specification of real-time systems have been proposed. Typical applications can be found in avionics, robotics, process control, and healthcare. For real-time applications, the meeting of temporal constraints are mandatory. A system specification must formalize system behavior (including temporal constraints) and, thus, the model must

be supported by mechanisms that verify conformity with requirements. Behavior is typically expressed by giving a set of relationships enumerating the temporal constraints among events and actions, such as invariants, precedence among events, periodicity, liveness and safety conditions, etc. – [Bucci et al. 1995], [Stoyenko 1992], [Stankovic 1988], [Stankovic and Ramamritham 1990]. The specification techniques must be formal enough to verify and validate the specification with respect to system requirements by using theorem provers or model checking techniques.

To this end, many researchers have proposed logical languages integrating temporal logics — e.g., [Pnueli 1977], [Jahanian and Mok 1986], [Schwartz et al. 1983], [Gotzhein 1992], [Vila 1994], [Orgun and Ma 1994]. These languages, together with several algebraic languages augmented with time (Z++, VDM++, Object Z, etc.) – e.g., [Zave 1982], [Lano 1991], [Lano and Haughton 1994], [Carrington et al. 1990], [Dürr and vanKatwijk 1992] — provide the most abstract approaches to requirement specification and real-time system analysis [Ostroff 1992], [Bucci et al. 1995]. Only in a few cases, logic specifications can be used for real implementation of the system.

The temporal logics proposed in the literature differ from each other with regard to expressiveness, availability of support tools for executability, verifiability, etc. In most cases, these temporal logics have been defined to satisfy specific needs. In recent years, the structure and capabilities of temporal logics has grown. In some cases, simple temporal logics are preferable to more complex and powerful ones, since the first are more satisfactorily adopted in certain applications. In order to clarify the differences among the several temporal logics, we reviewed a selection of the most representative temporal logics. Our review was based on building a taxonomy that classifies temporal logics in terms of order, time structure, decidability, executability, and expressiveness.

This paper reviews a number of well-known temporal logics designed for the specification of both reactive and/or real-time systems, taking into account their evolution in the last years and their expressiveness. The main features of temporal logics are discussed in view of their adoption for the specification of real-time systems. The features chosen for discussion include aspects of logic theory and applicability as specification languages. The above and other aspects are discussed in the paper together with several examples. Where possible, the same example has been used with several formalisms.

Classification criterion have been one of the major concerns of this paper. The world of temporal logics is surely far from stable, and we may have overlooked some relevant issues. Thus, we do not claim that this paper is an exhaustive review of temporal logics for the specification of real-time systems; rather it presents a useful taxonomy for classifying and identifying the capabilities of temporal logics, and can be used to classify even those temporal logics that may be proposed in the next years.

This paper is organized as follows. Section 2 discusses classical logics in order to highlight their limitations in expressing temporal properties. Section 3 highlights the most important features that characterize the temporal logics and presents some examples. Section 4 briefly describes the temporal logics selected and their suitability for the specification of real-time systems. Section 5 reports on the main features of the temporal logics considered. Conclusions are drawn in Section 6.

2. FROM CLASSICAL TO TEMPORAL LOGICS

The primary feature of a logic theory is its order, which defines the domain of all formulas described by the logic: (i) propositional, (ii) first order, (ii) higher order.

Formulae in *propositional logic* are built on the basis of a set of elementary facts (i.e., *atomic formulae*) by using a set of logic operators ($\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$). Their semantics can be defined in terms of *truth tables* or by inductive rules on the structure of the formula itself. Each formula can assume a logical value true ($\top$) or false ($\bot$).

The *First Order Logic*, FOL, adds several extensions to the propositional logic:

—There exists a domain of elements, $\mathcal{D}$, on the basis of which the logical formulae are built;

—n-ary $\mathcal{R}_i$ relationships on $\mathcal{D}$ can be defined, as subsets of $\mathcal{D}^n$;

—A n-ary predicate p_i is associated with each n-ary $\mathcal{R}_i$ relationship. The predicate is a function that for each element of $\mathcal{D}^n$ gives a value $\top$, if it belongs to an n-ary $\mathcal{R}_i$ relationship, otherwise the value $\bot$ is assumed;

—The operators of FOL are those of propositional logic plus the universal quantifier $\forall$ (*for all*), and the existential quantifier $\exists$ (*exists*).

In FOL quantified variables must be elements of $\mathcal{D}$ and not full predicates. The presence of quantification increases the expressiveness of the logic, allowing the description of existential and generalization relationships.

The *Higher Order Logic*, HOL, extends the domain modeled by FOL by allowing the adoption of predicates as quantification variables. For example, the following HOL formula:

$$\forall P.\exists x.P(x),$$

cannot be written in FOL since it contains a quantifier varying over a predicate P. The higher expressiveness of HOL makes it suitable for formally describing lower order logics.

2.1 Deductive Systems

Classical logics can formalize the deductive process: given a set of true propositions, it is possible to verify if other propositions are a logical consequence of the early set.

Proving theorems by using formal logic is a process quite different from the human deductive process. *Deductive systems* are based on a formalized theory by means of a set of axioms and deduction rules. This makes possible to define a purely syntax-deductive system without adopting the concepts of validity and satisfiability, which are typical of the human deductive process.

In order to profitably adopt a deductive system for proving theorems it is mandatory to demonstrate that it is *complete* (i.e., it is possible the construction of a demonstration for all theorems of the theory), and *sound* (i.e., each theorem that can be demonstrable with the logic is a theorem of the logic) [Davis et al. 1989], [Abramsky et al. 1992], [Ben-Ari 1993], [Andrews 1986].

Deductive systems tend to be minimal, as the set of axioms and deductive laws selected are usually just those strictly needed to describe the logic. Therefore,

the process required to prove other theorems can be complex and long. On the other hand, it is often possible to use the deductive system to demonstrate new deduction laws and theorems that, in turn, can be used in other proof processes as the minimum initial set. In this way, a system of deduction laws that makes the process of proof easier can be built.

2.2 Classical Logic and Time

In general, assertions can be classified as either static or dynamic. Static assertions have a fixed and time-independent truth value, while the truth value of dynamic assertions is in some way time-dependent. For example, the proposition $1 < 2$ is always true, whereas the logical value of the proposition:

it is raining

is time-varying: sometimes it may be true while at others it may be false. Since the state of real system changes over time, logic predicates describing the behavior must provide propositions whose values vary over time. Classical logic can express only *atemporal* (non-time dependent) formulas whose validity and satisfiability do not depend on the instant in which they are evaluated. In other words, time has no role in classical logic; when a proposition presents a value that changes over time, the time must be modeled as an explicit variable. For example, if a proposition P has to be true in interval $[t+5, t+20]$ we have to write the formula as

$$\forall x \in [t+5, t+20].P(x).$$

This approach makes the writing of time-dependent propositions quite complex. In order to model the behavior of domains in which the logical value of propositions may vary, modal and temporal logics were introduced as extensions of classical logic. These approaches facilitate the specification of temporal relationships.

2.3 Modal Logic

In modal logic, the concepts of truth and falsity are not static and immutable, but are, on the contrary, relative and variable [Hughes and Cresswell 1968]. In modal logic, the classical concept of interpretation of a formula is extended, in the sense that every modal logic theory has associated with it, not just a single interpretation, but a set of interpretations called *worlds*. In each world, a truth value is assigned to the formulas, similarly to the interpretation of a formula in classical logic.

A modal logic system is defined by $< W, R, V >$ where: W is the set of worlds; $R \subseteq W \times W$ is the *reachability* relationship between worlds; and V is the evaluation function for formulas:

$$V : F \times W \rightarrow \{\top, \bot\},$$

where F is the set of the formulas of the modal theory. V assigns a truth value to every formula in F in every world in W.

The forms of W and V depend on other characteristics of the logic; for example, whether it is propositional or a FOL. Besides the operators and symbols of classical logic, modal logic introduces operators $\boldsymbol{L}$ (*necessary*) and $\boldsymbol{M}$ (*possibly*). These express the concept of the necessity and possibility of formulas in the set of worlds reachable from the world in which the main formula is evaluated.

The semantics of a modal logic can be formally given on the basis of an evaluation function V, which is inductively defined over the structure of the formula to be evaluated. Omitting the definition of the part about classical logic, V is defined over the modal operators $\boldsymbol{L}$ and $\boldsymbol{M}$ as follows:

—$V(\boldsymbol{M}f, w) = \top$ iff $\exists v \in W.wRv \Rightarrow V(f, v)$;
—$V(\boldsymbol{L}f, w) = \top$ iff $\forall v \in W.wRv \Rightarrow V(f, v)$.

In other words, formula $\boldsymbol{M}f$ is true in a world w if and only if there exists a world v reachable from w, where subformula f is true; formula $\boldsymbol{L}f$ is true in w if and only if in all worlds reachable from w subformula f is true. Modal operators $\boldsymbol{L}$ and $\boldsymbol{M}$ have a simple interpretation as quantifiers defined over the set of reachable worlds from the current world, namely: $\boldsymbol{M}$ is an existential quantifier, while $\boldsymbol{L}$ is a universal quantifier. It is easy to see that the following relation holds between operators $\boldsymbol{L}$ and $\boldsymbol{M}$:

$$\boldsymbol{L}f = \neg\boldsymbol{M}\neg f.$$

The features of a modal logic $< W, R, V >$ are strictly connected to the relationship that determines the structure of the set of worlds. The interpretations of relationship R may be several: R can represent how a set of classical theories are correlated; for example, in a non-monotonic logic the elementary truth and the deducible facts can change dynamically. In the context of temporal logics the most interesting interpretation for relationship R is the relation *next instant*. In this way, the worlds are the set of configurations that the system modeled may assume in successive time instants. In this case, the modal logic can be quite profitably used for the study of temporal properties of systems, and for this reason takes the name *temporal logic*.

2.4 Temporal Logic

Temporal logics are particular modal logics where the set of worlds W is interpreted as the set of all possible instants T of a temporal domain.

Usually temporal logics are built as extensions of classical logic by adding a set of new operators that hide quantification over the temporal domain. Temporal logics presented in the literature are principally obtained by extending propositional or FOL; rarely has the extension started with HOL.

As in modal logic, where the world in which the formula is evaluated is referenced, in temporal logic the evaluation instant of a formula is used. The value of a formula is a *dynamic concept*. Therefore, the concept of formula satisfiability must be modified to consider both the interpretation of a formula and the instant of the evaluation.

Generally temporal logics add four new operators with respect to classical logics [Prior 1967]:

—$\boldsymbol{G}$, always in the future;
—$\boldsymbol{F}$, eventually in the future;
—$\boldsymbol{H}$, always in the past;
—$\boldsymbol{P}$, eventually in the past.

These can be formally defined:

—$V(\boldsymbol{G}f,t) = \top$ iff $\forall s \in T.t < s \Rightarrow V(f,s)$;
—$V(\boldsymbol{H}f,t) = \top$ iff $\forall s \in T.s < t \Rightarrow V(f,s)$;
—$\boldsymbol{F}f \equiv \neg\boldsymbol{G}\neg f$;
—$\boldsymbol{P}f \equiv \neg\boldsymbol{H}\neg f$.

These operators can express the concepts of necessity ($\boldsymbol{G}$, $\boldsymbol{H}$) and possibility ($\boldsymbol{F}$, $\boldsymbol{P}$) in the future and in the past, respectively. Often in temporal logics these operators are represented by other symbols: $\Box$ (*always*) denotes $\boldsymbol{G}$ and $\Diamond$ (*eventually*) denotes $\boldsymbol{F}$. For past operators (if they are present), symbol $\blacksquare$ denotes $\boldsymbol{H}$ and $\blacklozenge$ denotes $\boldsymbol{P}$.

If relation $<$ is transitive and non-reflexive, it is possible to introduce two other binary operators:

—**until** (in some cases represented with $\mathcal{U}$), with ϕ_1**until** ϕ_2 that is true if ϕ_2 will be true in the future and until that instant ϕ_1 will be always true;
—**since** (in some cases represented with $\mathcal{S}$), with ϕ_1**since** ϕ_2 that is true if ϕ_2 was true in the past and since that instant ϕ_1 has been true;

The semantic of these operators can be formally defined as follow:

—$V(f_1\textbf{until}f_2,t) = \top$ iff $\exists s \in T.t < s \wedge V(f_2,s) \wedge \forall u \in T.t < u < s \Rightarrow V(f_1,u)$;
—$V(f_1\textbf{since}f_2,t) = \top$ iff $\exists s \in T.s < t \wedge V(f_2,s) \wedge \forall u \in T.s < u < t \Rightarrow V(f_1,u)$.

Note that operator **until** (**since**) does not include the present instant in the future (past). The introduction of operators **until** and **since** is relevant since these operators can express concepts that cannot be expressed with the operators $\boldsymbol{G}$, $\boldsymbol{H}$, $\boldsymbol{F}$ and $\boldsymbol{P}$. On the contrary, these last operators can be defined in terms of **until** and **since**:

—$\boldsymbol{F}\phi \equiv \top\textbf{until}\phi$;
—$\boldsymbol{P}\phi \equiv \top\textbf{since}\phi$;

and

—$\boldsymbol{G}\phi \equiv \neg\boldsymbol{F}\neg\phi$;
—$\boldsymbol{H}\phi \equiv \neg\boldsymbol{P}\neg\phi$.

If the temporal logic has the *begin* property (e.g., stating that the temporal domain is bounded in the past as discussed in the sequel), the operator **until** is enough to complete the logic expressiveness: when the past is limited the operator **since** is not necessary. Relationships among events in the past can be expressed by using **until** starting from the beginning of time (from a fixed reference time instant).

Other common operators are *next* and *prev*, represented with $\bigcirc$ and $\bullet$, respectively. These operators are unary and can be defined in term of *until* and *since* operators:

—$\bigcirc\phi \equiv \bot\textbf{until}\phi$
—$\bullet\phi \equiv \bot\textbf{since}\phi$

These two operators assume different meanings depending on the time structure – e.g., discrete or continuous – or whether the logic is event-based.

The presence of distinct operators for past and future simplifies the specification model: since with their use formulas can be easily written – for instance, evaluating the past and describing the future. On the other hand, this distinction is only a convention, since in most temporal logics formulas can be easily shifted to the past or to the future.

3. MAIN CHARACTERISTICS OF TEMPORAL LOGICS

This section presents the evaluation criteria used to compare the temporal logics discussed in the following sections. We provide a taxonomy to classify and evaluate the suitability of temporal logics used for specifying real-time systems. Temporal logics are typically used in the phases of *requirements analysis*, *advanced analysis*, *specification*, and more recently, even for *execution*. They focus on modeling system behavior rather that functional or structural aspects [Bucci et al. 1995]. *Structural* aspect refers to system decomposition into subsystems (modular temporal logics). *Functional* aspect deals with the data transformation of the system. *Behavior* refers to the system reaction to external stimuli and internal events, a critical aspect of reactive and real-time systems.

To use temporal logics for real-time system specification, it is necessary to evaluate their expressiveness in modeling the typical requirements of such systems and of the constraints needed to express the specification. Typical temporal constraints can be divided in two main categories: (i) events and event orderings; (ii) quantitative temporal constraints.

The following paragraphs discuss the most important features of temporal logics and the criteria used to identify their general characteristics and properties.

3.1 Order of Temporal Logic

The order of a temporal logic is the order of classical logic on which the temporal logics is constructed. This characteristic dictates the set of formulas that the temporal logic can express. A higher order implies greater expressiveness but more complex formulas, and frequently, the logic itself is less complete and decidable. For instance, propositional temporal logics are less expressive than higher order logics, but often propositional temporal logics are decidable and their decision procedures have a tractable complexity; whereas higher order logics are more expressive but much more complex. First order temporal logics usually permit one to write quite expressive formulas without overly increasing the complexity of the logic.

3.2 Temporal Domain Structure

As stated in Section 2.3, the main properties of a modal logic, and then of a temporal logic, are related to the properties of relation R; the next section will show the structure of temporal domains derived from properties of relationship R. For temporal logics relation R is called a *precedence relation* and is denoted by $<$. Properties that bear on the temporal domain structure are:

transitivity	$\forall xyz.x < y \wedge y < z \Rightarrow x < z$
non-reflexivity	$\forall x.\neg x < x$
linearity	$\forall xy.x < y \vee x = y \vee y < x$
left linearity	$\forall xyz.y < x \wedge z < x \Rightarrow y < z \vee y = z \vee z < y$
right linearity	$\forall xyz.x < y \wedge x < z \Rightarrow y < z \vee y = z \vee z < y$
begin	$\exists x.\neg\exists y.y < x$
end	$\exists x.\neg\exists y.x < y$
predecessor	$\forall x.\exists y.y < x$
successor	$\forall x.\exists y.x < y$
density	$\forall xy.x < y \Rightarrow \exists z.x < z < y$
discreteness	$(\forall xy.x < y \Rightarrow \exists z.x < z \wedge \neg\exists u.x < u < z) \wedge$ $(\forall xy.x < y \Rightarrow \exists z.z < y \wedge \neg\exists u.z < u < y)$

Usually $<$ is a transitive and non-reflexive relationship; hence it is a partial ordering on time instants.

The property *begin* (*end*) states that the temporal domain is bounded in the past (future) [Halpern et al. 1983], [MelliarSmith 1987], whereas the property *predecessor* (*successor*) shows that the temporal domain is unlimited in the past (future). In fact, the following equivalencies hold:

$$(\exists x.\neg\exists y.y < x) \Leftrightarrow \neg(\forall x.\exists y.y < x)$$
$$(\exists x.\neg\exists y.x < y) \Leftrightarrow \neg(\forall x.\exists y.x < y)$$

A temporal domain is *dense* with respect to relationship $<$ if between two instants there is always a third. On the contrary, the temporal domain is *discrete* if there exist two instants between which a third cannot be determined.

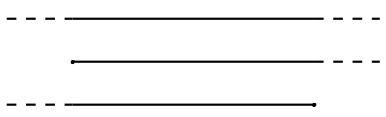

Fig. 1. Representation of linear temporal domains.

If the precedence relation is *linear* then we have a *linear temporal structure* that corresponds to the intuitive notion of time. This is the simplest type of temporal structure. In this case, the precedence relation is a total order on time instants. Figure 1 shows the temporal domain for a linear temporal structure with a unlimited past and future, with only a unlimited future and with only an unlimited past. If the time structure is linear and discrete, a state of the system can be associated with each time instant. If the time is dense, the logic must be event-based to support a state-based semantics.

When the precedence relation, $<$, is only *linear* on the left, the temporal structure is more complex: branches can exist in the future (in other words, more than one future can exist for each instant), but there exists only one past (see Figure 2). If the time is discrete and its structure is branched a next state exists but it cannot be unequivocally determined.

Temporal structures with branches in the past are also possible. If no hypotheses are made about linearity, branches in the future and in the past are possible.

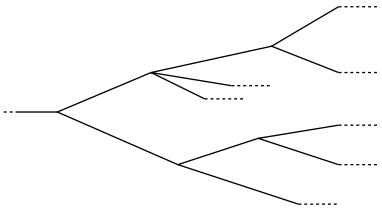

Fig. 2. A non linear structure in the future.

The order relation of the structure is usually transitive and non-reflexive. The temporal domain may be limited in the past and/or in the future or unlimited, and it may be dense or discrete. Thus, the temporal structure may be linear or branched in the past and/or in the future. These properties have implications for the decidability of the logic, its executability, and the style used to write formulas.

3.3 Fundamental Entity of the Logic

A basic way to characterize temporal logics is whether points or intervals are used to model time. This also influences the expressiveness of the logic.

Point-based temporal logics express relationships among events in terms of points. Point-based logics define intervals as connected set of points. In point-based logics it is more difficult to express relationships between intervals in which certain events are verified. Time durations are expressed by using quantifications over time. Logics based on *time points* [Manna and Pnueli 1983], [Rosner and Pnueli 1986] specify system behavior with respect to certain reference points in time; points are determined by a specific state of the system and by the occurrence of events marking state transition. In order to describe temporal relationships, the operators $\Box$ (*henceforth*) and $\Diamond$ (*eventually*) are usually adopted to specify *necessity* and *possibility*, respectively.

Interval-based temporal logics (interval logics) are more expressive since they are capable of describing events in time intervals and a single time instant is represented with a time interval of one. Usually interval-based logics permit one to write formulas with a greater level of abstraction and so are more concise and easy to understand than point-based temporal logics. In the case of *time intervals* [Schwartz and Melliar-Smith 1982], [Schwartz et al. 1983], [Moszkowski 1986], [Halpern et al. 1983], [Halpern and Shoham 1986], [Ladkin 1987], [MelliarSmith 1987] [Razouk and Gorlick 1989], formulæ specify the temporal relationships among facts, events, and intervals, thus allowing a higher level of abstraction for system specification. Interval-based logics usually present specific operators to express the relationships between intervals (*meet*, *before*, *after* [Allen 1983]), and/or operators for combining intervals (e.g., the *chop* operator [Rosner and Pnueli 1986]), or operators to specify the interval boundaries on the basis of the truth of predicates [MelliarSmith 1987].

The qualitative relationships that may hold between intervals as classified by Allen in [Allen and Ferguson 1994] are represented in Figure 3.

The relationships among time points or intervals are typically qualitative, but quantitative temporal logics are preferable for the specification of real-time systems (e.g., RTL [Jahanian and Mok 1986], MTL [Koymans 1990] and TRIO [Ghezzi et al.

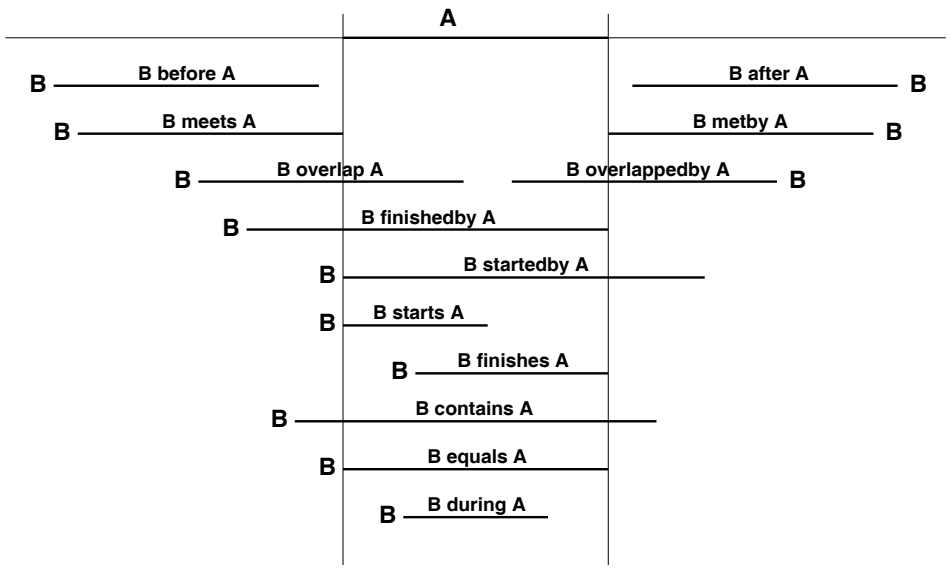

Fig. 3. Possible relationships between two intervals.

1990], TILCO [Mattolini and Nesi 1996]).

3.4 A Metric for Time and Quantitative Temporal Constraints

The presence of a metric for time determines the possibility of expressing temporal constraints in a quantitative form in the logic formulas; without a metric for time only temporal-order relations can be expressed (qualitative temporal logics).

The temporal operators presented in Section 2.4 are qualitative since it is not possible to give an exact measure (i.e., duration, timeout) for events and among events. Temporal logics without a metric for time adopt a time model for which the events are those that describe the system evolution (event-based temporal logics). Each formula expresses what the system does at each event, events are referred to other events, and so on: this results in specifying relationships of precedence and cause-effect among events.

Temporal logics with a metric for time allow the definition of quantitative temporal relationships, – such as distance among events and durations of events, in time units. The expression of quantitative temporal constraints is fundamental for real-time systems specification. It is necessary to have a metric for time if the temporal logic has to be used to express the behavior of hard or non-hard real-time systems. A typical way for adding a metric for time is to allow the definition of bounded operators – for example:

$$\Diamond_{[4,7]} A$$

for stating that A is eventually true from 4 to 7 time instants from the current time, or $\Diamond_{\leq 5} A$ which means that A is eventually true within 5 time units. A different method is based on the explicit adoption of a general system clock in the formulas

(see section 3.6).

A different way to manage time quantitatively is to adopt the *freeze* quantifier [Alur 1992], which allows only references to times that are associated with states. This means that freeze quantifier "$x.$" differs from the FOL quantification over time. For instance:

$$\Box x.\ (p \rightarrow \Diamond y.\ (q \vee y \leq x + 6))$$

This means that in every state with time x, if p holds, then there is a future state with time y such that q holds and y is at most $x + 6$. Logics allowing *freeze* quantification are called half-order logics.

In specifying of real-time systems, the general behavior of the system is typically expressed by means of quantitative temporal constraints. The correct behavior of the system depends on the satisfiability of these temporal constrains.

In [Koymans 1990], a classification of temporal constraints with respect to event occurrences has been proposed. In particular, we can specify constraints for establishing relationships between the occurrence of

(1) an event and a corresponding reaction (**reaction time**). Typical cases are:
—maximum distance between event and reaction (e.g., *timeout*);
—exact distance between event and reaction (e.g., *delay*);

(2) the same event (**period**). Typical cases are:
—minimum distance between two occurrences of an event;
—exact distance between occurrences of an event.

This classification can be simplified by reducing the types of temporal constraints to only two elementary constraints:

—universal temporal quantifier $\Box_i A$, that means that A is true in all time instants of interval i;

—existential temporal quantifier $\Diamond_i A$, that means that A is true in at least one time instant in interval i;

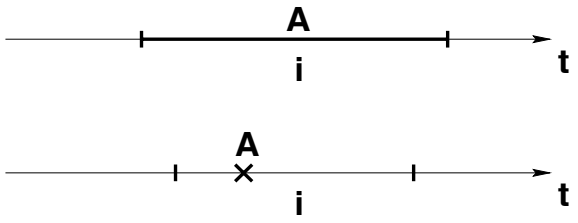

Fig. 4. Quantitative temporal constraints.

where A is a temporal logic formula and i is an interval that can be either a set of points or a fundamental entity whose extremes are expressed quantitatively (see Figure 4). By using these two elementary operators most of the possible temporal requirements of real-time systems can be expressed. For example:

—when A happens, B must happen within t time units: $A \Rightarrow \Diamond_{[0,t)} B$
—when A happens, B must happen after t time units: $A \Rightarrow \Box_{[t,t]} B$
—the distance between two occurrences of event A is at least t time units: $A \Rightarrow \Box_{(0,t)} \neg A$
—the distance between two occurrences of event A is always equal to t time units:

$$A \Rightarrow (\Box_{(0,t)} \neg A) \wedge (\Box_{[t,t]} A)$$

where intervals are specified using the usual mathematical notation, with round and squared brackets used for excluding and including bounds, respectively. The intervals are defined relative to the instant in which formulas are evaluated, so the time is implicit.

The above two elementary temporal operators are sufficient for expressing *safeness* or *liveness*. For example, the classical safety conditions, such as $\Box_i A$ (where A is a positive property) must be satisfied by the system specification, where the interval i can be extended to the *specification temporal domain*, as well as to only a part of it. Liveness conditions, such as $\Diamond_i A$ (A will be satisfied within i) or deadlock-free conditions, such as $\Box_j (\Diamond_i \neg A)$ can also be specified.

If unbounded intervals are allowed operators $\Box$, $\Diamond$, $\blacksquare$ and $\blacklozenge$ can be defined as:

—$\Box\phi \equiv \Box_{(0,+\infty)} \phi$;
—$\Diamond\phi \equiv \Diamond_{(0,+\infty)} \phi$;
—$\blacksquare\phi \equiv \Box_{(-\infty,0)} \phi$.
—$\blacklozenge\phi \equiv \Diamond_{(-\infty,0)} \phi$.

Certain temporal logics also provide bounded versions of the operators **since** and **until**. These versions can be easily obtained from the unbounded operators **since** and **until** and the bounded operators **henceforth** and **always**.

Some other temporal logics are much more oriented towards presenting the behavior of predicates intended as signals. These logics have been frequently used for modeling digital signals and are typically based on intervals. In order to relate the definition of an interval for bounding predicates with the evolution of other predicates a special operator for capturing the time instant related to events is needed. This special function from $Predicate \rightarrow Time$ is frequently introduced by using special operators.

3.5 Events and Ordering of Events

Typical relationships of cause-effect can be specified by using the simple operators imply ($\Rightarrow$) and co-imply ($\Leftrightarrow$). Moreover, the simpler operators of temporal logic (i.e., $\Box$ and $\Diamond$) can be profitably used for describing facts and rules. A *fact* is a predicate that is true at least for a time (e.g., presence of an event), while a *rule* is a predicate that is true in all time instants. These operators are unsuitable for specifying relationships of ordering among events, such as:

(a). A precedes B;
(b). A follows B;
(c). A will be true until B will become true for the next time;
(d). A has been true since the last time that B was true for the last time;

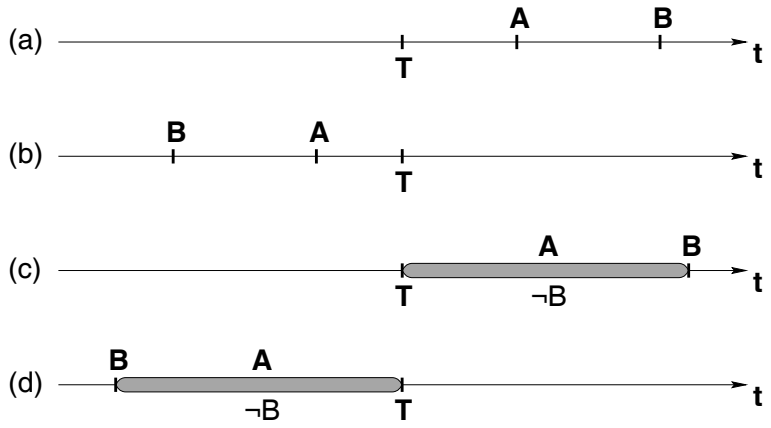

Fig. 5. Constraints about the ordering between events.

where A and B are temporal logic formulas. In Figure 5 graphical representations of (a) through (d) are shown, where **T** represents the instant in which formulas are evaluated. Constraints (c) and (d) may be described by using operators **until** and **since**, respectively. The precedence relation in the future (past) may be defined with operator **until** (**since**), as is shown by Manna in [Manna and Pnueli 1983], defining operators **precede** and **follow**:

$$A \textbf{ precede } B \equiv \neg((\neg A)\textbf{until}B)$$
$$A \textbf{ follow } B \equiv \neg((\neg A)\textbf{since}B).$$

Therefore, in order to express the ordering between events the temporal logic has to provide the operators **until** and **since**.

In effect, several versions of until and since operators exist. The typical definition of the until/since operator is the "weak" definition:

—A **until**$_w$ B – is true if B will become true and until that instant A will be true, or if B will stay always false and A always true.

—A **since**$_w$ B – is true if B has been true since the instant in which A became true, or if B has been always false and A always true.

The strong version of these operators assumes the occurrence of the change of status for B. Therefore, they can be defined in terms of the above operators as follows:

—A **until** $B \equiv \Diamond B \wedge A$ **until**$_w$ B

—A **since** $B \equiv \blacklozenge B \wedge A$ **since**$_w$ B

Different versions can be defined, and the current time can also be included in the evaluation range of the operators. In this case, the so-called 0 version of the weak version of the operators can be defined as follows:

—A **until**$_{w0}$ $B \equiv B \vee (A \wedge A$ **until**$_w$ $B)$

—A **since**$_{w0}$ $B \equiv B \vee (A \wedge A$ **since**$_w$ $B)$

Other versions can be defined for combinations of the basic versions stated above.

3.6 Time Implicit, Explicit, Absolute

Time in temporal logics can be defined in an *implicit* or *explicit* manner. A time model is implicit when the meaning of formulas depends on the evaluation time, and this is left implicit in the formula. For instance, $\Box A$ means that:

$$\forall t \in [T_0, \infty].A(t)$$

where T_0 is the evaluation time (the so-called current time instant). When time is implicit, the formalism is able to represent the temporal ordering of events. Each formula represents what happens in the evaluation time (e.g., in the past or in the future of the evaluation time), which is the implicit current time:

$$\Diamond \Box_{[3,5]} A$$

means that A will be eventually true in the future for an interval of 3 to 5 time units later with respect to the evaluation time. If time is treated implicitly, the possibility of referring the specification to an absolute value of time is usually lost. Temporal logics with time implicit may or may not allow the quantification over time (e.g., TRIO allows quantification over time and adopts a implicit model of time).

On the contrary, when the time is explicit the language represents the time through a variable. In this way, it is possible to express any useful property of real-time. The explicit specification of time allows the specification of expressions that have no sense in the time domain – e.g., the activation of a predicate when the time is even.

The reference to time can be *absolute* or *relative*. It is considered absolute when the value of the current time is referenced to a general system clock (the clock is idealized in the sense that no drift is supposed). It is frequently represented with T; for example, in the following formula an absolute explicit model of time is used:

$$\forall t.\Box(E \wedge T = t) \rightarrow \Diamond(A \wedge T - t < 10ms)$$

where E is an event. When time is expressed in absolute form, time durations and deadlines are given directly in seconds or milliseconds (i.e., the absolute time on the clock). Therefore, the meeting of timing constraints depends on the context (machine type, number of processes, workload, etc.).

The formula that follows has a relative explicit model of time:

$$\forall t.\Box(E \wedge T = t) \rightarrow \Diamond(A \wedge T - t < 10)$$

Frequently, time is expressed in a relative manner – that is, time durations and deadlines are given in time units. In this case, the relationship between these time units and the absolute measure of time expressed in seconds (or milliseconds) is left until the implementation phase. However, the validation of specifications becomes almost implementation independent. A different definition for absolute and relative time has been reported in [Koymans 1990].

3.7 Logic Decidability

The decidability of a temporal logic is related to the concepts of validity and satisfiability. A formula is satisfiable if there exists an interpretation for the symbols in the formula for which the formula is true, whereas a formula is valid if for every interpretation the formula is true. This feature is strongly related to the order of the logic. First-order (discrete time) temporal logic is incomplete, and validity and satisfiability problems are undecidable in the general case. This is mainly due to the quantification of time dependent variables. The prohibition of this kind of quantification has often been shown to be a necessary condition for the existence of feasible automated verification mechanisms such as in TPTL [Alur and Henzinger 1990].

Satisfiability (validity) is a decidable problem for a logic if there exists a decision procedure for the satisfiability (validity) of every formula of the logic. If one of these problems is decidable for the logic then the proof of theorems may be automatic. This property is highly desirable because it increases the logic's usability, since automatic instruments to verify and validate specifications can be built. This property is much more useful for temporal logics that are based on property proofs for the verification and validation of system properties. The adoption of a theorem prover confers an absolute certainty about the behavior of the system.

Other temporal logics have a semantics defined in terms of state evolution. This makes their application much more operational than descriptive [Bucci et al. 1995]. For these models, verification and validation activities are typically performed by using model-checking techniques. Unfortunately, for real systems, the verification of the system behavior in all its states can be infeasible because it is too complex and time consuming, even using symbolic model-checking algorithms. A semantics based on state is frequently associated with the presence of an event-based temporal logic or of a discrete linear model of time. In both these cases, the definition of an operational semantics for the temporal logic is quite simple.

3.8 Deductive System *sound* and *complete*

As expressed in Section 2.1, a deductive system is a formalization of the deduction process that is usually used to make proofs manually. A deductive system permits one to build proofs manually in simpler way and provides the basis for automating some simple rewriting of formulas. These mechanisms are typically used in automatic and semiautomatic theorem provers. Naturally it must be proved that this deductive system is *sound*, so that all proofs built are correct.

Another desirable but less "necessary" property, is the completeness of the deductive system; that is, the capacity to build a proof for *every* theorem true for the logic. It should be noted that it is never possible to build a *complete* deductive system: for example, the theory of natural numbers on FOL is sound but not complete; that is, there are non-provable true formulas [Davis 1989].

3.9 Logic Specification Executability

The problem of executability of specifications given by means of temporal logics has often been misunderstood. This mainly depends on the meaning assigned to executability [Fisher and Owens 1995], [Moszkowski 1986], [Barringer et al. 1996]. There are at least three different definitions of executability, as follows.

(i). Specification models are considered to be executable if they have a semantics defining an effective procedure, capable of determining for any formula of the logic theory, whether or not that formula is a *theorem* of the theory [Moszkowski 1986]. In effect, this property corresponds to that of decidability of the validity problem rather than to that of system specification executability.

(ii). A second meaning refers to the possibility of generating a model for a given specification [Felder and Morzenti 1992]. A detailed version of this concept leads to verifying if an *off-line* generated temporal evolution of inputs and outputs is compatible with the specification. This operation is usually called history checking.

(iii). The last meaning for executability consists of using the system specification itself as a prototype or implementation of the real-time system, thus allowing, in each time instant, the *on-line* generation of system outputs on the basis of present inputs and its internal state and past history. When this is possible, the specification can be directly executed instead of traducing it in a programming language.

In the literature, there exist only few executable temporal logics that can be used to build a system prototype according to meaning *(iii)* of executability. In general, the execution or simulation of logic specifications with the intent of producing system outputs in the correct time order by meeting the temporal constraints is a quite difficult problem. The difficulty mainly depends on the computational complexity of the algorithms proposed.

Moreover, while executing propositional temporal logics is a complex task, executing first order temporal logics is undecidable and highly complex [Fisher and Owens 1995], [Merz 1995]. A solution for executing propositional temporal logics could be (a) to restrict the logic and providing an execution algorithm for the remaining part, or (b) to execute the complete logic by using specific inferential rules and/or backtracking techniques. For first order temporal logics the solution can be to apply the same approaches used for propositional temporal logics or to try to build a model for the formula as in *(i)* and *(ii)* above.

If a temporal logic is executable the system can be simulated and/or executed. Thus, it is possible to validate system behavior through simulation and to use the system specification as a prototype or as an implementation if the execution speed is high enough to satisfy temporal constraints of the system.

4. A SELECTION OF TEMPORAL LOGICS

This section presents a selection of the most interesting types of temporal logics for the specification of real time systems. There are many other temporal logics in the literature, but most of them can be regarded as generalizations or specializations of those discussed here in.

The order in which the logics are presented is quite close to the chronological, from the earliest to the latest, from the simplest to its more complex evolutions (if present). Several examples are given in order to make the comparisons among the temporal logics presented possible. The section concludes with a brief discussion of the logics and a table for comparison purposes.

4.1 PTL: Propositional Temporal Logic

The Propositional Temporal Logic (PTL) introduced by Pnueli [Pnueli 1977], [Pnueli 1981], [Pnueli 1986] (see also [Ben-Ari 1993]), extends the propositional logic introducing temporal operators $\Box$, $\Diamond$, $\bigcirc$, and $\mathcal{U}$. The propositions of PTL describe temporal relationships between states that characterize the temporal evolution of the system. PTL is an event-based logic and does not provide a metric for time.

System requirements are specified by describing a set of constraints on the event sequences that occur in the system modifying its state. Time consists of a sequence of instants corresponding with the sequence of states of the system. In a certain sense, the fundamental entity of the logic is the instant in which the state of the system changes. For these reasons it is particularly suitable for integration in operational models such as state machines [Bucci et al. 1995].

The temporal structure of PTL is linear, bounded in the past (an initial instant exists), unbounded in the future (an infinite sequence of future states exists) and discrete (i.e., the set of instants is modeled with the set of natural numbers). For this reason, only temporal operators in the future are present. The temporal operators $\Box$, $\Diamond$ and $\mathcal{U}$ correspond to the operators $\boldsymbol{G}$, $\boldsymbol{F}$ and **until** described in Section 2.4. The formula $\bigcirc\phi$ is a valid formula if the formula ϕ is true in the *next* state. Operator **until** in PTL is equivalent to $\mathbf{until}_0$ presented in Section 3.5. Since PTL provides the operator **until** it is possible to specify real-time system requirements about the order of events in the future. The asymmetry of the logic (due to the boundary in the past) and the absence of the operator **since** does not permit specification of requirements about the order of events in the past. Moreover, the absence of a metric for time does not allow specification of any type of quantitative temporal constraint. Therefore, PTL is much more suitable for use with reactive and concurrent systems than with real-time systems. Reactive systems are typically event-driven and do not present quantitative temporal constraints such as timeouts or deadlines.

PTL is decidable (for example using a decision procedure based on the semantic tables method) and it is possible to build a *sound* and *complete* deductive system for the logic. In the literature, methods or instruments for executing PTL formulas have not been presented and, in general, these formulas are not executable.

In [Manna and Pnueli 1990], Manna and Pnueli proved that for an extension of PTL built adding symmetric operators in the past for $\blacksquare$, $\blacklozenge$, $\bullet$ and $\mathcal{S}$ it is possible to transform formulas of a particular class in finite state machines, thus permitting the execution of some formulas of this extension of PTL.

Table 1 shows some examples of the extended version of PTL. The table also shows a set of specifications that cannot be expressed by using this temporal logic. In the next subsections, similar tables are provided to allow comparison of the several temporal logics on the basis of a collection of equivalent specifications.

In [Barringer et al. 1989], [Barringer et al. 1991], [Finger et al. 1993], METAMEM is presented. METATEM includes an executable model and algorithm and can be considered to be based on an extended version of PTL.

Table 1. Some specifications in extended PTL.

meaning	PTL
Always A in the Past	$\blacklozenge A$
Always A in the Future	$\Box A$
Always A	$\blacksquare A \wedge \Box A$
A Since Weak B	$(A \mathcal{S} B \vee \blacksquare A)$
A Until Weak B	$(A \mathcal{U} B \vee \Box A)$
Lasts A up to t_1	–
Lasted A from $-t_1$	–
A Within $-t_1$ in the Past	–
A Within t_1 in the Future	–
A Within $(-t_1, t_2)$	–
A Was true in $(-t_1, -t_2)$	–
A Will be true in (t_1, t_2)	–
A Could be true in (t_1, t_2)	–
A Since B during $(-t_1, -t_2)$	–
A Until B during (t_1, t_2)	–

4.2 Choppy Logic

The Choppy Logic presented by Rosner and Pnueli [Rosner and Pnueli 1986] is an extension of PTL obtained by adding operator $\mathcal{C}$ (*chop*). This logic has all characteristics of PTL and enhances its expressiveness with operator $\mathcal{C}$, that permits one to concatenate state sequences. In the first approximation, the Chop operator can be regarded as an operator for dividing time intervals. In particular, a state sequence σ is a model for formula $\phi \, \mathcal{C} \psi$ if it can be divided in two sequences σ' and σ'' such that: σ' is a model for ϕ, and σ'' is a model for ψ. This logic has a greater expressiveness than PTL, but a more complex decision procedure is required. Thus, the Choppy Logic maintains all merits and problems of PTL.

4.3 BTTL: Branching Time Temporal Logic

The Branching Time Temporal Logic (BTTL) introduced by Ben-Ari, Pnueli and Manna [Ben-Ari et al. 1983] is an extension of PTL. It has a temporal structure with branches in the future, and thus could be used for describing the behavior of non-deterministic systems. PTL operators are enhanced to deal with branches. Four operators have been defined to quantify both on different evolution traces and states that are present on the selected traces:

—$\forall\Box$, for all traces π and for all states $s \in \pi$;

—$\exists\Box$, for at least one trace π and for all states $s \in \pi$;

—$\forall\Diamond$, for all traces π and at least one state $s \in \pi$;

—$\exists\Diamond$, for at least one trace π and for at least one state $s \in \pi$.

In several aspects BTTL is practically equivalent to PTL. Moreover, it adopts a temporal structure branched in the future. BTTL also presents a *complete* axiomatization; it is decidable; and the satisfiability of formulas can be determined by using a method based on semantic tables that also produces models for the BTTL formulas. The models for the formulas are finite and could be used to build finite state machines corresponding to the formulas. This makes the model operationally

executable. Even with this improvement of PTL it is not possible to specify quantitative temporal constraints. Thus, this logic is also not suitable for real-time systems specification.

4.4 ITL: Interval Temporal Logic

The Interval Temporal Logic (ITL) introduced by Halpern, Manna and Moszkowski [Halpern et al. 1983] and used/further studied by Moszkowski in [Moszkowski and Manna 1984], [Moszkowski 1985], [Moszkowski 1986] can be considered as an extension of PTL. ITL is a propositional logic with a temporal structure that is bounded in the past, unbounded in the future, discrete and linear. The fundamental entity of ITL is the interval made of a sequence of states. The length of an interval is defined as the number of states in the sequence. ITL does not provide a metric for time and can be considered an event-based logic. It has been applied for modeling the evolution of digital signals. ITL extends the propositional logic with the operators $\bigcirc$ (*next*), $\Box$, $\Diamond$ and ";" (*chop*, analogous to operator $\mathcal{C}$ of Choppy Logic). The semantics of all these operators is defined in terms of intervals rather then of states as in PTL. From the above basic operators a set of derived operators has been defined. The presence of operator *chop* makes the satisfiability of ITL formulas undecidable; nevertheless, the satisfiability is decidable for a particular subclass of ITL formulas. It is possible to build a *sound* deductive system for ITL.

In [Moszkowski 1986] Tempura is presented. It is a subset of ITL formulas with some syntactic properties, for which the problem of building an execution for formula is tractable, even if unsolvable in the general case. In ITL only order properties showing qualitative relationships among the order of events can be specified. This makes this logic less powerful for specifying real-time systems. To specify order properties the operator *chop* must be used since ITL does not have operator **until**. As a surrogate of the metric for time a special operator $Len(n)$ is used to count the number of states in a sequence. This allows one to specify the exact duration in terms of number of transitions among events.

4.5 PMLTI: Propositional Modal Logic of Time Intervals

The Propositional Modal Logic of Time Intervals (PMLTI) presented by Halpern and Shoham [Halpern and Shoham 1986] is a temporal logic that extends the propositional logic. The fundamental temporal entity is the interval and the temporal operators can express the possible relationships between intervals, as reported in Figure 3. The temporal structure requires only the total order of the points in the intervals. With this limitation, the time structure can be linear or branched, bounded or unbounded, dense or discrete. PMLTI does not provide an explicit metric for time. The selection of a specific temporal structure leads to implications about the complexity of the decision procedure for demonstrating the validity of formulas. The problem of validity and satisfiability of PMLTI formulas may be decidable or undecidable depending on the temporal structure chosen.

PMLTI uses a method of translating temporal logic formulas in FOL formulas of a specific deductive system to proof theorems of the logic. This approach enables application of all the techniques which are available for first order logic. To date, the problem of formula executability has not been addressed. The presence of operators for the specification of relationships between intervals permits one to

easily express event order constraints. However, the absence of a metric for time makes the expression of quantitative temporal constraints impossible.

4.6 CTL: Computational Tree Logic

The Computational Tree Logic (CTL) presented by Clarke, Emerson and Sistla [Clarke et al. 1986], [Clarke and Grumberg 1987], [Stirling 1987] is a propositional branching time temporal logic. The fundamental temporal entity is the point and presents specific operators for reasoning about the system behavior in terms of several futures, called sequences. It is very similar to BTTL. CTL does not provide an explicit metric for time. For verifying CTL specification a model-checking approach is typically used since the specification can be modeled as a state machine [Clarke et al. 1986]. In [Emerson and Halpern 1986], [Emerson et al. 1989] a real-time extension of CTL has been presented, RTCTL, presenting a metric for time. The satisfiability problem for this logic is doubly exponential. The model-checking has a polynomial time algorithm, [Ostroff 1992]. In [Josko 1987], a modular version of CTL has been presented, MCTL.

4.7 IL: Interval Logic

The Interval Logic (IL) presented by Schwartz, Melliar-Smith and Vogt [Schwartz et al. 1983], [Schwartz and Melliar-Smith 1982] is based on time interval and propositional logic. The temporal structure is linear, bounded in the past and unbounded in the future. IL does not present an explicit metric for time. Time intervals are bounded by events and by the changes of system state described by the formulas. Therefore, IL is an event-based logic. A typical IL formula is in the following form:

$$[\mathcal{I}]\alpha,$$

where α is a formula and $\mathcal{I}$ is the interval that is the context of which formula α has to be verified. This formula means that the next time the interval can be built then the formula α will hold in it. The most interesting feature of IL is the set of instruments that can be used for the determination and construction of time intervals. It presents bounded versions of operators $\Diamond$ and $\Box$. The bound is defined by means of the interval: $[\mathcal{I}]\Diamond\alpha$ means that α can be true in $\mathcal{I}$. The interval bounds can be defined by occurrence of events. Given an interval the initial and final intervals can be extracted. Moreover, the existence of an interval with certain characteristics is an event. Finally, to describe system behavior operators *at*, *in* and *after* have been defined; these specify the truth at the start, during and at the end of the interval, respectively. They may be used as events for construction of intervals. For instance: $A \Rightarrow$ as interval means that the interval starts when A starts and ends at the end of the context.

Results for testing the executability of IL do not exist. This is because IL has been introduced as a specification language and is verified by means of automatic instruments, without taking into consideration the possibility of simulating or executing the specifications.

IL permits one to easily write constraints about the order of events using the instruments for the construction of context intervals, but it cannot be used to specify quantitative temporal constraints, as can the extensions discussed in subsections 4.8 and 4.9 below.

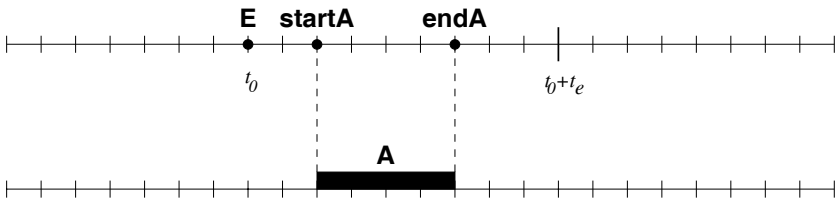

Fig. 6. One example for several temporal logics.

4.8 EIL: Extended Interval Logic

Extended Interval Logic (EIL) was introduced by Melliar-Smith [MelliarSmith 1987]. It extends IL by adding the possibility of specifying some types of quantitative temporal constraints. These extensions have been introduced to eliminate the incapacity of IL to express the typical requirements of real-time systems. The first extension is the possibility of defining an event from another event at a constant temporal distance (positive or negative): if E is an event then $E + 1$sec is also an event. The second extension is the possibility of limiting the length of intervals. For example: formula < 2sec is true if the interval in which it is evaluated has a duration of less than 2 seconds, while > 10min is true if the interval has a duration greater than 10 minutes. The extensions introduced add the capability of expressing some of the quantitative temporal constraints that are needed to specify real-time systems. For instance:

$$\Box[E \Rightarrow *endA](< t_e \wedge *startA)$$

means that for each occurrence of event E predicates *startA* and *endA* (marking an interval in which A is true) hold and this interval is included from the occurrence of the E and t_e (see Figure 6, in which t_0 is time instant in which E occurs). In the above formula operator $*$ can be read as *exists an occurrence of*, while $\Rightarrow$ means that the left bound of the interval is defined by the occurrence of event E.

4.9 RTIL: Real-Time Interval Logic

The Real-Time Interval Logic (RTIL) presented by Razouk and Gorlick [Razouk and Gorlick 1989] is another extension of IL. In this case the goal was to permit the specification of real-time systems with the specific intention of verifying the consistency between the execution traces and the system specification itself. RTIL extends IL by introducing a metric for time. It can assign a temporal value to the extremes of the intervals and can construct intervals by assigning numerical values at interval bounds, not only by using events and state changes. Moreover, it is possible to measure the interval duration. This characteristic makes RTIL interesting for the specification of real-time systems. For example, the specification described in Figure 6 can be written as:

$$\Box[\odot E \hookrightarrow t_e] * (\odot\ startA \Rightarrow \odot\ endA)$$

In this case, operator $*$ has to be read as *exists a subinterval*. The special operator $\odot A$ extracts the time instant in which A becomes true. *endA* and *startA* have the same meanings as in EIL. Instants can be specified absolutely or relative to

the beginning of the current context. RTIL also permits quantification over finite domains. This feature does not enhance the expressiveness of the logic but simplifies the writing of complex and repetitive formulas.

4.10 LTI: Logic of Time Intervals

The Logic of Time Intervals (LTI) of Allen [Allen and Ferguson 1994] is an interval temporal logic of the second order. It is also called Interval Time Logic (acronym ITL). To avoid confusion with the ITL presented it will be referred to in this paper as LTI. Intervals can be divided in subintervals. Intervals that cannot be further divided into subintervals constitute *moments*. The logic permits quantification of temporal intervals. The temporal structure is linear, without any further limitations – even the model of time can be either discrete or dense. LTI does not provide an explicit metric for time. Temporal propositions are made by declaring the order relationships between intervals (see Figure 3). In [Ladkin 1987], it has been shown that LTI theory is incomplete and proposes a way to make it complete. Furthermore, it is shown that both the theories, the new and complete, and the early and incomplete versions, are decidable. An axiomatic system is provided for both, although there are not known results about logic executability. LTI does not present problems for ordering constraints regarding the expression of the typical temporal constraints of real-time systems. Specification of quantitative temporal constraints is impossible since the measure of the length of intervals is missing.

4.11 RTTL: Real-Time Temporal Logic

The Real-Time Temporal Logic (RTTL) presented by Ostroff and Wonham [Ostroff and Wonham 1987], [Ostroff 1989], [Ostroff and Wonham 1990], [Ostroff 1992] extends PTL with proof rules for real-time properties. The temporal structure is linear and discrete; the fundamental entity is the point. Time is limited in the past and unlimited in the future. Time is defined with both a sequence of state and a sequence of temporal instants. The presence of a state-based model makes RTTL particularly suitable for model-checking techniques; thus it can be used as a model to verify small systems. A natural number is associated with each time instant; thus, RTTL is based on an explicit model of time. The *clock* of the system is periodically incremented and it is accessible for writing formulas. State changes can occur: (i) corresponding with the changes of time of system, or (ii) between two successive instants. In the case in which more events occur between two successive instants, these events are distinguishable only for the order in which they occur, and not for the temporal instant associated with the occurrences. For this reason, the metric for time is only partial: non-simultaneous events that occur for the same value of the system clock may exist. Operator **until** of PTL and operator **until** of RTTL are equivalent to $\mathbf{until}_0$ presented in Section 3.5. In RTTL, quantification of *rigid* variables is allowed. Rigid variables are variable in the set of possible executions but are constant for each execution. RTTL is a first order logic. For RTTL it is essential that the system clock (T) value be referenced in formulas to express some types of concepts, such as to establish relationships between different temporal contexts. Table 2 shows some RTTL specifications; no specifications involving the past are shown since RTTL presents only the future.

All global variables (e.g., t in the table) in formulas are assumed to be universally

Table 2. Some specifications in RTTL.

meaning	RTTL
Always A in the Past	–
Always A in the Future	$\bigcirc\Box A$
Always A	–
A Since Weak B	–
A Until Weak B	$\bigcirc(A\mathcal{U}B)\vee\Box A$
Lasts A up to t_1	$t=T\rightarrow\Box((t<T\wedge T<t+t_1)\rightarrow A)$
Lasted A from $-t_1$	–
A Within $-t_1$ in the Past	–
A Within t_1 in the Future	$t=T\rightarrow\Diamond((t<T\wedge T<t+t_1)\wedge A)$
A Within $(-t_1,t_2)$	–
A Was true in $(-t_1,-t_2)$	–
A Will be true in (t_1,t_2)	$t=T\rightarrow\Box((t+t_1<T\wedge T<t+t_2)\rightarrow A)$
A Could be true in (t_1,t_2)	$t=T\rightarrow\Diamond((t+t_1<T\wedge T<t+t_2)\wedge A)$
A Since B during $(-t_1,-t_2)$	–
A Until B during (t_1,t_2)	$t=T\rightarrow A\mathcal{U}(B\wedge t+t_1<T\wedge T<t+t_2)$

quantified [Ostroff 1991]. The logic also presents **next** operator $\bigcirc$. "Lasts A up to t_1" can be also written in a more concise notation $\Diamond_{(0,t_1)}A$ while "A Until B during (t_1,t_2)" can be specified as $A\mathcal{U}_{(t_1,t_2)}B$. The situation described in Figure 6 can be specified by using:

$$\Box\left(E\rightarrow\bigcirc\left((\Diamond_{\leq t_e}\ endA)\wedge\neg(\neg startA\,\mathcal{U}\ endA)\right)\right)$$

considering predicates *startA* and *endA* as above. Note the adoption of bounded operator $\Diamond$. The possibility of adopting (i) an explicit reference to the system clock value, and (ii) indirect quantifications on values assumed by the clock leads to the ability to write every type of ordering and quantitative constraints (the above example is implicitly quantified on t). This is extremely interesting for the specification of real-time systems. However, this flexibility leads to the production of formulas that are quite difficult to understand and manipulate with respect to other temporal logics that avoid quantification over time-dependent variables. A *sound* deductive system has been built for RTTL (extending a deductive system of PTL), but the satisfiability problem is undecidable. The suitability of RTTL for model checking and the presence of a deductive system makes RTTL a dual model according to the classification reported in [Bucci et al. 1995]. No results about the executability of RTTL specification are available. TTM/RTTL is a dual approach obtained by the integration of a state machine model and RTTL [Ostroff and Wonham 1987], [Ostroff 1989], [Ostroff and Wonham 1990]. TTM is an operational model based on communicating finite state machines in which variables with arbitrary domains are used. The *operations* allowed are variable assignment, send, and/or receive. The state machine follows a Mealy model in which conditions on transitions between states are equivalent to logic formulae on state variables, while the output is an assignment to state variables.

4.12 TPTL: Timed Propositional Temporal Logic

In [Alur and Henzinger 1989], Alur and Henzinger presented the Timed Propositional Temporal Logic (TPTL) and in [Alur and Henzinger 1990] they have shown

the expressiveness and complexity of this logic. TPTL is an extension of PTL. Like PTL, TPTL is a propositional logic, where the instant is the fundamental temporal entity and the time is linear, discrete, limited in the past, unlimited in the future. An extension with respect to PTL is the presence of a metric for time: every instant corresponds to a natural number and a monotone function associates a temporal value with each state of the system, thus making timed state sequences possible. The presence of operator **until** permits one to specify order constraints. The possibility of specifying quantitative temporal constraints is one of the fundamental characteristics of the logic. For these reasons, this logic is suitable for specifying real-time systems requirements. Its theoretic bases that facilitate requirement verification and validation. Table 3 shows some specifications in TPTL. No specifications in the past are shown since TPTL presents only the future. TPTL adopts the *freeze* operator, thus x and y represent time instants. The specifications are quite similar to RTTL. Adoption of *freeze* operator can be very interesting to model system in which more than a real-time clock is present. A typical application is the specification of communicating systems in which distinct specifications have to be synchronized (see APTL in [Wang et al. 1993]).

Table 3. Some specifications in TPTL.

meaning	TPTL
Always A in the Past	–
Always A in the Future	$\bigcirc\Box A$
Always A	–
A Since Weak B	–
A Until Weak B	$\bigcirc(\mathcal{U}BA \vee \Box A)$
Lasts A up to t_1	$x.\Box y.(x < y < x + t_1) \rightarrow A$
Lasted A from $-t_1$	–
A Within $-t_1$ in the Past	–
A Within t_1 in the Future	$x.\Diamond y.(x < y < x + t_1) \wedge A$
A Within $(-t_1, t_2)$	–
A Was true in $(-t_1, -t_2)$	–
A Will be true in (t_1, t_2)	$x.\Box y.(x + t_1 < y < x + t_2) \rightarrow A$
A Could be true in (t_1, t_2)	$x.\Diamond y.(x + t_1 < y < x + t_2) \wedge A$
A Since B during $(-t_1, -t_2)$	–
A Until B during (t_1, t_2)	$x.\bigcirc\mathcal{U}\,(y.B \wedge (x + t_1 < y < x + t_2))\,A$

The situation described in Figure 6 is specified in TPTL by using:

$$\Box x.E \rightarrow (\Diamond y.\ endA \wedge y \leq x + t_e) \wedge \neg(\mathcal{U}\ endA\ \neg startA)$$

considering predicates *startA* and *endA* as above. In [Alur and Henzinger 1990], it has been proven that the choice of the set of natural numbers for a temporal domain is essential to obtaining a temporal logic for which the satisfiability problem is decidable. In fact, for every temporal domain with a more complex structure than natural numbers, the problem of satisfiability is undecidable. PTL's deductive systems can be extended and transformed for TPTL by retaining the properties of *soundness* and *completeness*. Moreover, a decision procedure based on the semantic

table algorithm and a model-checking algorithm has been presented. This facilitates the use of this logic for the specification and verification of real-time systems requirements.

4.13 RTL: Real-Time Logic

The Real-Time Logic of Jahanian and Mok [Jahanian and Mok 1986] is a logic that extends the first-order logic with a set of elements for the specification of real-time systems requirements. RTL proposes a logic approach for the specification of real-time systems, but is not a temporal logic in the classical meaning. It presents an absolute clock to measure time progression. The value of this clock can be referenced in the formulas: function "@" permits one to assign a temporal value (execution instant) to an event occurrence. The temporal domain is the set of natural numbers, and is linear, discrete limited in the past, unlimited in the future, and totally ordered. The fundamental entity is the time instant. In RTL, there are no problems in specifying ordering and quantitative temporal constraints, since it is possible to make explicit reference to time even through quantification. The main problem with RTL is the fact that absolute system time is referenced, with a low level of abstraction, leading to very complex formulas required to describe the system. The example of Figure 6 is specified in RTL by using:

$$\forall t.\forall i.@(\Omega E, i) = t \rightarrow (\exists j.(t \leq @(\uparrow A, j)) \wedge (@(\downarrow A, j) \leq t + t_e))$$

Operator ΩE states the occurrence of external event E; $\uparrow A$ the turning true from false of predicate/signal A; $\downarrow A$ the becoming false from true of A; i and j are the occurrences of the events marked with operator @; t is the time. Note the need of a quantification over time to specify the example. In [Alur and Henzinger 1990], it has been shown that RTL is undecidable even when the syntax is restricted. In [Jahanian and Mok 1986] a procedure to demonstrate the consistency of *safeness* assertions relative to real-time system specification is proposed. A deductive system for RTL has not been presented, but it seems to be feasible by extending a system for FOL with laws for the new operators. There are no known results regarding the executability of RTL. In [Armstrong and Barroca 1996] an approach based on RTL and Statechart was presented. In that case, the formal verification was provided by using a theorem prover.

4.14 TRIO: Tempo Reale ImplicitO

TRIO is a logic language for real-time system specification (Tempo Reale ImplicitO - Implicit Real Time). It has been presented by Ghezzi, Mandrioli and Morzenti [Ghezzi et al. 1990], [Felder and Morzenti 1994]. TRIO extends FOL with specific predicates for real-time system specification. The temporal structure is linear and totally ordered: possible temporal domains are the natural numbers, the integers, the real numbers, or an interval of one of these set. The fundamental temporal entity is the point and a metric for time is available. On that basis, it is possible to measure the distance of two points and the length of an interval. Since TRIO is an extension of FOL, which is undecidable, then TRIO is also an undecidable logic. TRIO presents only two temporal operators: $\mathbf{Futr}(A, t)$ and $\mathbf{Past}(A, t)$ for specifying that A occurs at time instant t in the future and past, respectively (more recently it has been demonstrated that both these operators can be defined

in terms of a unique operator). Moreover, in TRIO, based on these operators, several other operators can be defined as parametric predicates. This is frequently allowed by many temporal logics - e.g., TILCO, MTL. The temporal operators introduced by TRIO, with the possibility of quantification on temporal variables without any restriction, permit the expression of order and quantitative temporal constraints as needed for real-time systems specification. It is necessary to use quantification over the time domain, so formulas are often complex and difficult to read and manipulate. Table 4 shows some specifications in TRIO, the table presents three columns. The middle column shows the specification written on the basis of TRIO's elementary operators, while the column on the right shows the version of the specification in a derived form. This derived form can be obtained by defining a new temporal operator (special parameterized predicate) with the specification reported in the middle column or by using already defined operators. It is possible to define new "temporal operators" by means of special functions: on the one hand, this keeps the size of formulas low, but on the other hand, it makes the language harder to understand. A large number of operators can create confusion during the specification process, especially when these specifications have to be understood by other analysts who do not know the definitions of the same predicates implementing complex temporal operators.

Table 4. Some TRIO simple temporal specifications.

meaning	TRIO	TRIO derived
Always Past	$\forall t(t > 0 \rightarrow \mathbf{Past}(A, t))$	AlwP(A)
Always Future	$\forall t(t > 0 \rightarrow \mathbf{Futr}(A, t))$	AlwF(A)
Always	$\forall t(t > 0 \rightarrow \mathbf{Futr}(A, t)) \wedge A \wedge \forall t(t > 0 \rightarrow \mathbf{Past}(A, t))$	Alw(A)
Since Weak	$\forall t''(t'' > 0 \rightarrow \mathbf{Past}(A, t''))\vee$	
	$\exists t(t > 0 \wedge \mathbf{Past}(B, t) \wedge \forall t'(0 < t' < t \rightarrow \mathbf{Past}(A, t)))$	Since$_w(B, A)$
Until Weak	$\forall t''(t'' > 0 \rightarrow \mathbf{Futr}(A, t''))\vee$	
	$\exists t(t > 0 \wedge \mathbf{Futr}(B, t) \wedge \forall t'(0 < t' < t \rightarrow \mathbf{Futr}(A, t)))$	Until$_w(B, A)$
Lasts	$\forall t'(0 < t' < t \rightarrow \mathbf{Futr}(A, t'))$	Lasts(A, t)
Lasted	$\forall t'(0 < t' < t \rightarrow \mathbf{Past}(A, t'))$	Lasted(A, t)
Within Past	$\exists t'(0 < t' < t \wedge \mathbf{Past}(A, t'))$	WithinP(A, t)
Within Future	$\exists t'(0 < t' < t \wedge \mathbf{Futr}(A, t'))$	WithinF(A, t)
Within	$\exists t'(0 < t' < t_1 \wedge \mathbf{Past}(A, t')) \vee \ A \vee$	
	$\exists t''(0 < t'' < t_2 \wedge \mathbf{Futr}(A, t''))$	Within(A, t_1, t_2)
Was	$\mathbf{Past}(\forall t'(0 < t' < t_1 - t_2 \rightarrow \mathbf{Futr}(A, t')), t_1)$	$\mathbf{Past}(\text{Lasts}(A, t_1 - t_2), t_1)$
Will be	$\mathbf{Futr}(\forall t'(0 < t' < t_2 - t_1 \rightarrow \mathbf{Futr}(A, t')), t_1)$	$\mathbf{Futr}(\text{Lasts}(A, t_2 - t_1), t_1)$
Could be	$\mathbf{Futr}(\neg\forall t'(0 < t' < t_2 - t_1 \rightarrow \mathbf{Futr}(\neg A, t')), t_1)$	$\mathbf{Futr}(\neg\text{Lasts}(\neg A, t_2 - t_1), t_1)$
A Since B	$\exists t(0 < t_2 < t < t_1) \wedge \mathbf{Past}(B, t)\wedge$	
during $(-t_1, -t_2)$	$\forall t'(0 < t' < t \rightarrow \mathbf{Past}(A, t'))$	Since$_B(B, A, t_1, t_2)$
A Until B	$\exists t(0 < t_1 < t < t_2) \wedge \mathbf{Futr}(B, t)\wedge$	
during (t_1, t_2)	$\forall t'(0 < t' < t \rightarrow \mathbf{Futr}(A, t'))$	Until$_B(B, A, t_1, t_2)$

For TRIO, the example of Figure 6 is obtained by using:

$$Alw\,(E \rightarrow \exists t\,((0 < t < t_e) \wedge \mathbf{Futr}(endA, t) \wedge WithinF(startA, t)))$$

In this case, the specification has been obtained by using a user-defined operator $WithinF()$; its definition is provided in Table 4. Even in this case quantification

over time is needed. The same specification could be given by using operator *until* without the adoption of the quantification over time:

$$Alw\,(E \rightarrow WithinF(endA, t_e) \wedge \neg Until(endA, \neg startA))$$

A deductive system for TRIO has been presented. This system has been used to prove theorems for TRIO and to build a deductive system for Timed Petri Nets. TRIO has been used mainly for the validation and verification of system requirements through testing activity (history checking) and not by means of the proof of system properties. TRIO has been described as an executable logic language in the general sense. It can be used to build a model of the system under specification as TRIO formulas. Histories of system variables can be checked against the specification in order to verify whether they satisfy the specification. Therefore, TRIO must be considered a specific case of model checking and not a full execution according to the classification of [Fisher and Owens 1995].

4.15 MTL: Metric Temporal Logic

In [Koymans 1990] Koymans presented Metric Temporal Logic (MTL) that extends FOL with temporal operators from modal logic:$\boldsymbol{G}$, $\boldsymbol{F}$, $\boldsymbol{H}$, $\boldsymbol{P}$. MTL includes a metric for time according to some properties that describe the structure of the temporal domain. One of these properties states that the order of the temporal structure has to be total, thus leading to a linear temporal structure. The fundamental entity of the logic is the temporal point. The presence of the metric for time permits one to modify the temporal operators making temporal versions of most of the above-discussed temporal operators: $\boldsymbol{G}$, $\boldsymbol{F}$, $\boldsymbol{H}$, $\boldsymbol{P}$. This allows one to reduce the needs of using quantifications on temporal domain. The operators **until** and **since** can be obtained on the basis of the other operators as depicted in Table 5. These provide support for avoiding the adoption of quantification over time. In Table 5 some MTL specifications are given. MTL presents both past and future operators. The three columns in Table 5 have the same meaning as the table presented for TRIO (see Table 4).

The example shown in Figure 6 for MTL can be obtained by using:

$$E \rightarrow \exists t\,(0 \lll t \lll t_e \wedge \mathbf{F}_t\, endA \wedge \mathbf{F}_{<t}\, startA)$$

The same specification could be written without the adoption of the quantification over time

$$E \rightarrow \mathbf{F}_{<te}\, endA \wedge \neg\,(\neg startA\ until\ endA)$$

As stated in [Alur and Henzinger 1990] MTL is undecidable, but a deductive system is available. The MTL operators permit one to specify constraints on event order (**until**, **since**) and quantitative temporal constraints ($\boldsymbol{G}$, $\boldsymbol{F}$, $\boldsymbol{H}$, $\boldsymbol{P}$). The executability of MTL has not been discussed in the literature.

4.16 TILCO: Time Interval Logic with Compositional Operators

In [Mattolini 1996], [Mattolini and Nesi 1999], and [Mattolini and Nesi 1996], Mattolini and Nesi presented TILCO (Time Interval Logic with Compositional Operators), a temporal logic for real-time system specification. TILCO extends the FOL and uses as a fundamental temporal entity the interval even if the interval is defined

Table 5. Some specifications in MTL.

meaning	MTL	MTL Derived
Always A in the Past	$\mathbf{H}A$	
Always A in the Future	$\mathbf{G}A$	
Always A	$\mathbf{H}A \wedge A \wedge \mathbf{G}A$	
A Since Weak B	$\mathbf{H}A \vee \exists t(t \gg 0 \wedge \mathbf{P}_t B \wedge \mathbf{H}_{<t} A)$	A *since* B
A Until Weak B	$\mathbf{G}A \vee \exists t(t \gg 0 \wedge \mathbf{F}_t B \wedge \mathbf{G}_{<t} A)$	A *until* B
Lasts A up to t	$\mathbf{G}_{<t} A$	
Lasted A from $-t$	$\mathbf{H}_{<t} A$	
A Within $-t$ in the Past	$\mathbf{P}_{<t} A$	
A Within t in the Future	$\mathbf{F}_{<t} A$	
A Within $(-t_1, t_2)$	$\mathbf{P}_{<t_1} A \wedge A \wedge \mathbf{F}_{<t_2} A$	
A Was true in $(-t_1, -t_2)$	$\mathbf{P}_{t_1}(\mathbf{H}_{<(t_1 - t_2)} A)$	
A Will be true in (t_1, t_2)	$\mathbf{F}_{t_1}(\mathbf{G}_{<(t_2 - t_1)} A)$	
A Could be true in (t_1, t_2)	$\mathbf{G}_{t_1}(\mathbf{F}_{<(t_2 - t_1)} A)$	
A Since B during $(-t_1, -t_2)$	$\exists t(t_2 \ll t \ll t_1 \wedge \mathbf{P}_t B \wedge \mathbf{H}_{<t} A)$	$\mathbf{P}_{t_2}(A\ \mathbf{since}_{<(t_1 - t_2)}\ B) \wedge \mathbf{H}_{<t_2} A$
A Until B during (t_1, t_2)	$\exists t(t_1 \ll t \ll t_2 \wedge \mathbf{F}_t B \wedge \mathbf{G}_{<t} A)$	$\mathbf{F}_{t_1}(A\ \mathbf{until}_{<(t_2 - t_1)}\ B) \wedge \mathbf{G}_{<t_1} A$

in terms of a couple of time instants. The temporal structure is linear and presents a metric for time that associates an integer number to every temporal instant; no explicit temporal quantification is allowed. In TILCO, the same formalism used for system specification is employed for describing high-level properties that should be satisfied by the system itself. These must be proven on the basis of the specification in the phase of system validation. Since TILCO operators quantify over intervals, instead of using time points, TILCO is more concise in expressing temporal constraints with time bounds, as is needed in specifying real-time systems. The basic temporal operators of TILCO are the existential and universal temporal quantifiers (**@** and **?**, respectively), and operators **until** and **since**). These operators permit a concise specification of temporal requirements, relationships of ordering and quantitative distance among events; thus TILCO fully supports the specification of real-time systems. TILCO is also characterized by its compositional operators that work with intervals: *comma* ",", which corresponds to $\wedge$, and *semicolon* ";", which corresponds to $\vee$, between intervals. Compositional operators "," and ";" assume different meanings if they are associated with operators "**@**" or "**?**":

$$\begin{aligned} A\mathbf{@}i, j &\equiv (A\mathbf{@}i) \wedge (A\mathbf{@}j), \\ A\mathbf{?}i, j &\equiv (A\mathbf{?}i) \wedge (A\mathbf{?}j), \\ A\mathbf{@}i; j &\equiv (A\mathbf{@}i) \vee (A\mathbf{@}j), \\ A\mathbf{?}i; j &\equiv (A\mathbf{?}i) \vee (A\mathbf{?}j). \end{aligned}$$

Other operators among intervals, such as intersection, "$\cap$", and union, "$\cup$", have been defined by considering time intervals as sets. Table 6 shows some specifications in TILCO. In this case, the table has only two columns; even in TILCO, special functions can be easily written for defining new temporal operators, such as in TRIO and MTL. However, in TILCO this is less necessary since TILCO specifications are quite concise, as can be noted by comparing Tables 4, 5, and 6.

For TILCO, the condition depicted in Figure 6 can be specified by using:

$$E \rightarrow\ endA\mathbf{?}(0, t_e] \wedge \neg\mathbf{until}(endA, \neg startA)$$

Table 6. Some specifications in TILCO.

meaning	TILCO
Always A in the Past	$A@(-\infty, 0)$
Always A in the Future	$A@(0, \infty)$
Always A	$A@(-\infty, \infty)$
A Since Weak B	$\mathbf{since}(B, A)$
A Until Weak B	$\mathbf{until}(B, A)$
Lasts A up to t	$A@(0, t)$
Lasted A from $-t$	$A@(-t, 0)$
A Within $-t$ in the Past	$A?(-t, 0)$
A Within t in the Future	$A?(0, t)$
A Within $(-t_1, t_2)$	$A?(-t_1, t_2)$
A Was true in $(-t_1, -t_2)$	$A@(-t_1, -t_2)$
A Will be true in (t_1, t_2)	$A@(t_1, t_2)$
A Could be true in (t_1, t_2)	$A?(t_1, t_2)$
A Since B during $(-t_1, -t_2)$	$B?(-t_1, -t_2) \wedge \mathbf{since}(B, A)@[-t_2, -t_2] \wedge A@[-t_2, 0)$
A Until B during (t_1, t_2)	$B?(t_1, t_2) \wedge \mathbf{until}(B, A)@[t_1, t_1] \wedge A@(0, t_1]$

In [Mattolini 1996] and [Mattolini and Nesi 1999] a sound deductive system for TILCO has been presented. This system is used in the context of the general theorem prover Isabelle [Paulson 1994] to provide an assisted support for proving TILCO formulas. Using this formalization, a set of fundamental theorems has been proven and a set of tactics has been built for supporting the semi-automatic demonstration of properties of TILCO specifications. Causal TILCO specifications are also executable by using a modified version of the Tableaux algorithm. Since TILCO has aspects typical of both descriptive and operational semantics, it can be considered a dual approach following the classification reported in [Bucci et al. 1995]. Since TILCO extends FOL, it is undecidable in the general case. However, the subset of formulas that presents only quantifications on finite sets is decidable. Causal TILCO specifications can be executed with a modified version of a tableaux algorithm.

5. DISCUSSION

In Table 7, the main characteristics of the temporal logics reviewed in the previous sections have been collected. The following discussion considers two main aspects of the logics: the intrinsic power of expressiveness in terms of logic order and quantification over time variable; and the readability/ understandability of the logics.

The temporal logics discussed can be divided into two main categories: temporal logics without a metric for time and those with a metric for time. PTL, Choppy Logic, BTTL, ITL, PMLTI, IL, CTL and LTI belong to the first category. These logics are less satisfactory for the specification of real-time systems since quantitative temporal constraints cannot be specified. In the second category, lie the following temporal logics: EIL, RTIL, RTTL, TPTL, RTL, TRIO, MTL, and TILCO. Some of these logics are characterized by the fact that they permit explicit quantification on the variable *time*, whereas for the others it is not permitted. In [Alur and Henzinger 1989], it has been observed that not permitting explicit quantification on time brings about a more natural specification style. Moreover, in

Table 7. Comparative table regarding the features of the temporal logics examined.

Logic	Logic order[1]	Fundamental time entity[2]	Temporal structure[3]	Metric for time/-Quantitative temporal constraints[4]	Logic decidability[4]	Deductive system[4]	Logic executability[4]	Ordering events[4]	Implicit, Explicit[5]
PTL	P	P	L	N	Y	Y	Y	Y	I
Choppy	P	P	L	N	Y	(Y)	(Y)	Y	I
BTTL	P	P	B	N	Y	Y	Y	Y	I
ITL	P	I	L	N	(Y)	(Y)	(Y)	Y	I
PMLTI	P	I	L/B	N	(Y)	NA	NA	Y	I
CTL	P	P	B	N	Y	NA	NA	Y	I
IL	P	I	L	N	Y	NA	NA	Y	I
EIL	P	I	L	Y	Y	NA	NA	Y	I
RTIL	P	I	L	Y	Y	NA	NA	Y	(I)
LTI	2^{nd}	I	L	N	Y	Y	NA	Y	(I)
RTTL	1^{st}	P	L	(Y)	N	Y	NA	Y	E
TPTL	P	P	L	Y	Y	Y	NA	Y	(E)
RTL	1^{st}	I	L	Y	N	NA	NA	Y	E
TRIO	1^{st}	P	L	Y	N	Y	(Y)	Y	I
MTL	1^{st}	P	L	Y	(N)	(Y)	NA	Y	I
TILCO	1^{st}	I	L	Y	(Y)	Y	(Y)	Y	I

[1] P= propositional, 1^{st} = first order, 2^{nd} = second order;
[2] P= point, I= interval;
[3] L= linear, B= branching;
[4] N= no, (N)=no in the general case, Y= yes, (Y)=yes in some specific case, NA= not available;
[5] I= implicit, E= esplicit.

[Alur and Henzinger 1990] the impossibility of explicit quantification on time was demonstrated to be a necessary condition for the existence of a practically usable verification method, such as the techniques based on tableaux. In fact, a logic that allows quantification over time has the expressive power of FOL and is undecidable. For this reason, in many cases, logics as EIL, RTIL, TPTL, and TILCO, are typically preferable to RTTL, RTL, TRIO, and MTL that permit quantifications over time. When a temporal logic allows the possibility of quantification on non-temporal variables (even with some limitations) it can be considered a first order temporal logic. This is a great advantage since it leads to a more expressive specification language and has a greater power of abstraction. Among the logics examined, only RTIL, RTL, TRIO, MTL, and TILCO permit quantification on non-time dependent variables. More specifically, only RTIL and TILCO seem to present the most complete collection of interesting characteristics for real-time systems specification (metric for time, expression of quantitative and events order temporal constraints, no quantification over time). Both of these logics do not permit quantification on time but permit the quantification on non-time dependent variables with finite domains. RTIL permits one only to reference the absolute time, and then only indirectly in a relative manner. Moreover, the order of events is not complete, since events having a relationship of *successor* or *predecessor* can occur for the same value of the system clock. TILCO does not have these problems and has a sound deductive system that supports the assisted proof of theorems and

execution of formulas.

From the point of view of readability and understandability of the temporal logic it is highly relevant to evaluate two aspects: the number of elementary operators, and the structure of the syntax. The first of these aspects is quite objective, since a lower number of temporal operators is typically preferred. Temporal logics that have a high number of operators are, like programming languages, typically hard to learn and hard to understand. Their expressiveness can be high, since a wide collection of operators or temporal predicates can be very useful for specifying complex systems, but ease of learning and readability are low. It has been previously shown that all the most useful specifications can be expressed by using a very low number of temporal operators. If these operators support a metric for time, their expressiveness is even higher. As a case limit, all the operators can be defined in terms of a measuring operator or modeled with delay. On the other hand, having too low a number of temporal operators can produce the same effects, since complex specifications have to be built by using elementary operators even for very simple specifications. This means that a balance between the power of the temporal logic and its number of temporal operators is needed. The number of operators also influences the syntax of the temporal logic. In some cases, the verbosity of temporal logic depends on the presence of a neat distinction between past and future – e.g., extended PTL, TRIO, MTL. This distinction typically leads to duplication of the number of operators in order to have specific operators for past and future. When this distinction is not made, time can considered only in the future – e.g., RTTL, TPTL – or more general and flexible operators capable of working continuously from past to future are defined – e.g., TILCO. In evaluating temporal logics, other interesting features can be the availability of (i) a graphical representation for the visual specification; (ii) a support for structuring communicating processes. The visual representation of temporal specifications has frequently been addressed by researchers who have neglected the capabilities of temporal logics. Visual representation may make the readability of the specifications easier, but their real expressiveness is given by the above-mentioned features of the temporal logics. An interesting integrated approach can be seen in [Dillon et al. 1994], [Moser et al. 1997]. The second aspect has been only marginally considered in this survey since, for many of the temporal logics presented, several researchers have discussed the possibility of specifying process/subsystem communication. These cases should be considered very carefully since the concept of communication directly implies the definition of a theory for supporting processes/modules. These can be processes (behavioral decomposition) or objects (structure decomposition). In any case, the complexity of these aspects cannot be described in few pages.

6. CONCLUSIONS

In this article, a series of criteria for assessing the capabilities of temporal logics for the specification, validation and verification of real-time systems have been presented. On the basis of the adopted criteria, most of the temporal logics examined have been found to be not-fully satisfactory for the specification of real-time systems. The criteria proposed delineate some essential characteristics that an "ideal" temporal logic should have to be profitably adopted for the specification of real-time

systems. This does not mean that temporal logics that do not have these features cannot be used for that purpose, but only that their adoption makes the specification of some temporal constraints hard and sometimes impossible. Frequently it happens that if a temporal constraint cannot be specified as imagined by the analysis, it may be specified in some other way by using different constructs. According to our point of view and to the trend of the temporal logics in recent years, the following features should be available to build a temporal logic strongly suitable for the specification of real-time systems: (i) based on FOL; (ii) prohibition of the quantification on time variables; (iii) presence of a metric for time; (iv) interval as fundamental time entity since the interval can be view as a generalization of the point; (v) relative time model and not absolute; (vi) a limited number of basic operators and the possibility of building special functions.

REFERENCES

Abramsky, S., Gabbay, D. M., and Maibaum, T. S. E. 1992. *Handbook of Logics in Computer Science, Vol.1*. Oxford Science Publications.

Allen, J. F. 1983. Maintaining knowledge about temporal intervals. *Communications of the ACM 26*, 11 (Nov.), 832–843.

Allen, J. F. and Ferguson, G. 1994. Actions and events in interval temporal logic. Technical report (July), University of Rochester Computer Science Department, TR-URCSD 521, Rochester, New York 14627.

Alur, R. 1992. Logics and models of real time: A survey. Technical report (Jan.), Dept. of Computer Science Cornell University, Ithaca, New York, USA.

Alur, R. and Henzinger, T. A. 1989. A really temporal logic. In *30th IEEE FOCS* (1989).

Alur, R. and Henzinger, T. A. 1990. Real time logics: complexity and expressiveness. In *Proc. of 5th Annual IEEE Symposium on Logic in Computer Science, LICS 90* (Philadelphia, USA, June 1990), pp. 390–401. IEEE. YES in Kavi92, TR STANCS901307, Dept. of Comp. Science and Med, Stanford.

Andrews, P. B. 1986. *An Introduction to Mathematical Logic and Type Theory: to Truth Through Proof*. Academic Press, Inc., Orlando, USA. YES, Mattolini.

Armstrong, J. and Barroca, L. 1996. Specification and verification of reactive system behavior: The railroad crossing example. *Journal of Real-Time Systems 10*, 143–178.

Barringer, H., Fisher, M., Gabbay, D., Gough, G., and Owens, R. 1989. Metatem: A framework for programming in temporal logic. In *in Proc. of REX Workshop on Stepwise Refinement of Distributed Systems: Models Formalism, Correctness* (Mook, Netherlands, June 1989). Springer Verlag, LNCS n.430.

Barringer, H., Fisher, M., Gabbay, D., and Hunter, A. 1991. Meta-reasoning in executable temporal logic. In *Proc. of the 2nd International Conference on Principles of Knowledge Representation and Reasoning (KR)* (Cambridge, Massachusetts, April 1991). Morgan Kaufmann.

Barringer, H., Fisher, M., Gabbay, D., Owens, R., and Reynolds, M. 1996. *The Imperative Future: Principles of Executable Temporal Logic*. Research Studies Press, LTD, John Wiley & Sons Inc., New York.

Ben-Ari, M. 1993. *Mathematical Logic for Computer Science*. Prentice Hall, New York.

Ben-Ari, M., Pnueli, A., and Manna, Z. 1983. The temporal logic of branching time. *Acta Informatica 20*.

Bucci, G., Campanai, M., and Nesi, P. 1995. Tools for specifying real-time systems. *Journal of Real-Time Systems 8*, 117–172.

Carrington, D., Duke, D., Duke, R., King, P., Rose, G., and Smith, G. 1990. Object-z: An object-oriented extension to z. In S. T. Voung Ed., *Formal Description Techniques*. Elsevier Science.

Clarke, E., Emerson, E., and Sistla, A. 1986. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM TOPLAS 8*, 2 (April), 244–263.

Clarke, E. M. and Grumberg, O. 1987. The model checking problem for concurrent systems with many similar processes. In B. Banieqbal, H. Barringer, and P. Pnueli Eds., *Proc. of the International Conference on Temporal Logic in Specification* (Altrincham, UK, 8-10 April 1987), pp. 188–201. Springer Verlag, LNCS n.398.

Davis, L. S., DeMenthon, D., Bestul, T., Harwood, D., Srinivasan, H. V., and Ziavras, S. 1989. Rambo - vision and planning on the connection machine. In *Proc. of 6th Scandinavian Conference on Image and Application, Oulu, Finland* (19-22 June 1989), pp. 1–14.

Davis, R. E. 1989. *Truth, Deduction, and Computation: Logic and Semantics for Computer Science*. Computer Science Press, New York. YES, Univ. Firenze, Dip. Sist. ed Inf., P. Nesi, G. Bucci.

Dillon, L. K., Kutty, G., Moser, L. E., Melliar-Smith, P. M., and Ramakrishna, Y. S. 1994. A graphical interval logic for specifying concurrent systems. *ACM Transactions on Software Engineering and Methodology 3*, 2 (April), 131–165.

Dürr, E. H. H. and vanKatwijk, J. 1992. Vdm++: A formal specification language for object-oriented designs. In G. Heeg, B. Mugnusson, and B. Meyer Eds., *Proc. of the International Conference on Technology of Object-Oriented Languages and Systems, TOOLS 7* (1992), pp. 63–78. Prentice-Hall.

Emerson, E. A. and Halpern, J. Y. 1986. Sometimes and not never revisited: on branching versus linear time temporal logic. *Journal of the ACM 33*, 1 (Jan.), 151–178.

Emerson, E. A., Mok, A. K., Sistla, A. P., and Srinivasan, J. 1989. Quantitative temporal reasoning. In *Proc of the Workshop on Finite-State Concurrency* (Grenoble, 1989).

Felder, M. and Morzenti, A. 1992. Validating real-time systems by history-checking TRIO specifications. In *Proc. of 14th International Conference on Software Engineering* (Melbourne, Australia, 11-15 May 1992), pp. 199–211. IEEE press, ACM.

Felder, M. and Morzenti, A. 1994. Validating real-time systems by hystory-checking trio specifications. *ACM Transactions on Software Engineering and Methodology 3*, 4 (Oct.), 308–339.

Finger, M., Fisher, M., and Owens, R. 1993. Metamem at work: Modelling reactive systems using executable temporal logic. In *Proc. of the 6th International Conference on Industrial and Engineering Applications if Artificial Intelligence and Expert Systems (IEA-AIE* (Edinburg, UK, June 1993). Gordon and Breach.

Fisher, M. and Owens, R. 1995. An introduction to executable modal and temporal logics. In M. Fisher and R. Owens Eds., *Executable Modal and Temporal Logics: Proceedings of the IJCAI '93 Workshop, Chamberry, France, August 1993* (1995), pp. 1–20. Lecture Notes in Artificial Intelligence, Springer Verlag LNCS 897.

Ghezzi, C., Mandrioli, D., and Morzenti, A. 1990. Trio, a logic language for executable specifications of real-time systems. *Journal of Systems and Software 12*, 2 (May), 107–123.

Gotzhein, R. 1992. Temporal logic and applications – a tutorial. *Computer Networks and ISDN Systems, North-Holland 24*, 203–218.

Halpern, J., Manna, Z., and Moszkowski, B. 1983. A hardware semantics based on temporal intervals. In J. Diaz Ed., *Proc. of the 10th Colloquium on Automata Languages and Programming, Lecture Notes in Computer Science LNCS N.154* (Barcellona, Spain, July 1983), pp. 278–291. Springer Verlag.

Halpern, J. Y. and Shoham, Y. 1986. A propositional modal logic of time intervals. In *Proc. of the 1st IEEE Symp. on Logic in Computer Science* (1986), pp. 274–292. IEEE Press.

Hughes, G. E. and Cresswell, M. J. 1968. *An Introduction to Modal Logic*. Methuen, London.

Jahanian, F. and Mok, A. K.-L. 1986. Safety analysis of timing properties in real-time systems. *IEEE Transactions on Software Engineering 12*, 9 (Sept.), 890–904.

Josko, B. 1987. MCTL: An extension of CTL for modular verification of concurrent systems. In B. Banieqbal, H. Barringer, and P. Pnueli Eds., *Proc. of the International Conference on Temporal Logic in Specification* (Altrincham, UK, 8-10 April 1987), pp. 165–187. Springer Verlag, LNCS n.398.

Koymans, R. 1990. Specifying real-time properties with metric temporal logic. *Real-Time Systems Journal 2*, 255–299.

Ladkin, P. 1987. Models of axioms for time intervals. In *Proc. of the 6th National Conference on Artificial Intelligence, AAAI'87* (USA, 1987), pp. 234–239.

Lano, K. 1991. Z++, an object-oriented extension to z. In J. E. Nicholls Ed., *Proc. of the 4th Annual Z User Meeting* (Oxford, UK, 1991), pp. 151–172. Workshop in Computing, Springer Verlag.

Lano, K. and Haughton, H. 1994. *Object-Oriented Specification Case Studies*. Prentice Hall, New York, London.

Manna, Z. and Pnueli, A. 1983. Proving precedence properties: The temporal way. In J. Diaz Ed., *Proc. of the 10th Colloquium on Automata Languages and Programming, Lecture Notes in Computer Science LNCS N.154* (Barcelona, Spain, July 1983), pp. 491–512. Springer Verlag.

Manna, Z. and Pnueli, A. 1990. A hierarchy of temporal properties. In *Proceedings of the 9th Symposium on Principles of Distributed Computing, Quebec, Canada, Aug. 1990* (1990), pp. 377–408.

Mattolini, R. 1996. *TILCO: a Temporal Logic for the Specification of Real-Time Systems (TILCO: una Logica Temporale per la Specifica di Sistemi di Tempo Reale)*. Ph. D. thesis.

Mattolini, R. and Nesi, P. 1996. Using tilco for specifying real-time systems. *Proc. of the 2nd IEEE Intl. Conference on Engineering of Complex Computer Systems, Montreal (Quebec, Canada)*, 18–25.

Mattolini, R. and Nesi, P. 1999. An interval logic for real-time system specification. *IEEE Transactions on Software Engineering, in press*.

MelliarSmith, P. M. 1987. Extending interval logic to real time systems. In *Proc. of Temporal Logic Specification United Kingdom, (B. Banieqbal, H. Barringer, A. Pnueli, eds)* (April 1987), pp. 224–242. Springer Verlag, Lecture Notes in Computer Sciences, LNCS 398.

Merz, S. 1995. Efficiently executable temporal logic programs. In M. Fisher and R. Owens Eds., *Executable Modal and Temporal Logics: Proceedings of the IJCAI '93 Workshop, Chamberry, France, August 1993* (1995), pp. 1–20. Lecture Notes in Artificial Intelligence, Springer Verlag LNCS 897.

Moser, L. E., Ramakrishna, Y. S., Kutty, G., Melliar-Smith, and Dillon, L. K. 1997. A graphical environment for the design of concurrent real-time systems. *ACM Transactions on Software Engineering and Methodology 6*, 1 (Jan.), 31–79.

Moszkowski, B. 1985. A temporal logic for multilevel reasoning about hardware. *Computer*, 10–19.

Moszkowski, B. and Manna, Z. 1984. Reasoning in interval logic. In *Proc. of ACM/NSF/ONR workshop on LOgics of Programs LNCS 164, Lecture Notes in Computer Science* (1984), pp. 371–384. Springer Verlag.

Moszkowski, B. C. 1986. *Executing Temporal Logic Programs*. Ph. D. thesis, Cambridge University press.

Orgun, M. and Ma, W. 1994. An overview of temporal logic programming. In *Proc. of the 1st International Conference on Temporal Logic (ICTL)* (Bonn, Germany, July 1994). Springer Verlag, LNCS N.827.

Ostroff, J. S. 1989. *Temporal Logic for Real-Time Systems*. Research Studies Press LTD., Advanced Software Development Series, 1, Taunton, Somerset, England. NO.

Ostroff, J. S. 1991. Verification of safety critical systems using ttm/rttl. In *Proc. of the REX Workshop on Real-Time: Theory in Practice, LNCS 600* (1991). Springer Verlag.

OSTROFF, J. S. 1992. Formal methods for the specification and design of real-time safety critical systems. *Journal of Systems and Software*, 33–60.

OSTROFF, J. S. AND WONHAM, W. 1987. Modeling and verifying real-time embedded computer systems. In *Proc. of the 8th IEEE Real-Time Systems Symposium*, pp. 124–132. IEEE Computer Society Press. NO.

OSTROFF, J. S. AND WONHAM, W. M. 1990. A framework for real-time discrete event control. *IEEE Transactions on Automatic Control 35*, 4 (April), 386–397. YES vedi Kavi92.

PAULSON, L. C. 1994. *Isabelle: A Generic Theorem Prover*. Lecture Notes in Computer Science, Springer Verlag LCNS 828.

PNUELI, A. 1977. The temporal logic of programs. In *18th IEEE FOCS* (1977). mattolini.

PNUELI, A. 1981. The temporal semantics of concurrent programs. *Theoretical Computer Science 13*.

PNUELI, A. 1986. Applications of temporal logic to the specification and verification of reactive systems: A survey of current trends. In *Current Trends in Concurrency*. Lecture Notes in Computer Science, Springer Verlag LNCS 224.

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

RAZOUK, R. R. AND GORLICK, M. M. 1989. A real-time interval logic for reasoning about executions of real-time programs. In *Proc. of the ACM/SIGSOFT'89, Tav.3* (Dec. 1989), pp. 10–19. ACM Press.

ROSNER, R. AND PNUELI, A. 1986. A choppy logic. In *Proc. of the 1st IEEE Symp. on Logic in Computer Science* (1986), pp. 306–313. IEEE Press.

SCHWARTZ, R. L. AND MELLIAR-SMITH, P. M. 1982. From state machines to temporal logic: Specification methods for protocol standards. *IEEE Transactions on Communications 30*, 12 (Dec.), 2486–2496.

SCHWARTZ, R. L., MELLIAR-SMITH, P. M., AND VOGT, F. H. 1983. A interval logic for higher-level temporal reasoning. In *Proc. of the 2nd Annual ACM Symp. Principles of Distributed Computing, Lecture Notes in Computer Science, LNCS N. 164* (Montreal Canada, 17-19 Aug. 1983), pp. 173–186. Springer Verlag, ACM NewYork.

STANKOVIC, J. A. 1988. Misconceptions about real-time computing: A serious problem for next-generation systems. *IEEE Computer*, 10–19.

STANKOVIC, J. A. AND RAMAMRITHAM, K. 1990. What is predictability for real-time systems. *Journal of Real-Time Systems 2*, 247–254.

STIRLING, C. 1987. Comparing linear and branching time temporal logics. In B. BANIEQBAL, H. BARRINGER, AND P. PNUELI Eds., *Proc. of the International Conference on Temporal Logic in Specification* (Altrincham, UK, 8-10 April 1987), pp. 1–20. Springer Verlag, LNCS n.398.

STOYENKO, A. D. 1992. The evolution and state-of-the-art of real-time languages. *Journal of Systems and Software*, 61–84.

VILA, L. 1994. A survey on temporal reasoning in artificial intelligence. *AI Communications 7*, 1 (March), 4–28.

WANG, F., MOK, A. K., AND EMERSON, A. 1993. Distributed real-time system specification. *ACM Transactions on Software Engineering and Methodology 2*, 4 (Oct.), 346–378.

ZAVE, P. 1982. An operational approach to requirements specification for embedded systems. *IEEE Transactions on Software Engineering 8*, 3 (May), 250–269.

Contents