An Interval Logic for Real-Time System Specification*

R. Mattolini (Member, IEEE), P. Nesi (Member, IEEE)
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@dsi.unifi.it, nesi@ingfi1.ing.unifi.it www: http://www.dsi.unifi.it/~nesi
email: mattolin@aguirre.ing.unifi.it

November 3, 1998

Abstract

Formal techniques for the specification of real-time systems must be capable of describing system behavior as a set of relationships expressing the temporal constraints among events and actions: properties of invariance, precedence, periodicity, liveness and safety conditions, etc. This paper describes a Temporal Interval Logic with Compositional Operators (TILCO) that has been expressly designed for the specification of real-time systems. TILCO can be regarded as a generalization of classical temporal logics based on the operators *eventually* and *henceforth* and allows both qualitative and quantitative specification of time relationships. TILCO is based on time intervals and can concisely express temporal constraints with time bounds, such as those needed to specify real-time systems. This approach can be used to verify the completeness and consistency of specifications, as well as to validate system behavior against its requirements and general properties. TILCO has been formalized by using the theorem prover Isabelle/HOL. TILCO specifications that satisfy certain properties are executable by using a modified version of the Tableaux algorithm. This paper defines TILCO and its axiomatization, highlights the tools available for proving properties of specifications and for their execution, and provides an example of system specification and validation.

Index terms: formal specification language, first order logic, temporal interval logic, verification and validation, real-time systems.

1 Introduction

In specifying real-time systems (avionics, robotics, process control, etc.) many factors must be considered. For example, the techniques adopted must be capable of describing system behavior as a set of relationships expressing the temporal constraints among events and actions [1], [2]: properties of invariance, precedence among events, periodicity, liveness and safety conditions, etc. Moreover, the specification techniques must be formal enough to allow verification and

*This work was partially supported by the Italian Research Council, CNR (Consiglio Nazionale delle Ricerche), n. 93.01865.CT/12.

validation of the specification with respect to system requirements and/or to real stimuli by using theorem provers or model-checking techniques.

During the last decade, many researchers have explored temporal representation for specifying concurrent as well as real-time programs. Many logical languages integrating constructs for temporal reasoning – temporal logics – have been proposed; e.g., [3], [4], [5], [6], [7]. These languages, together with algebraic languages — e.g., [8], [9], [10], [11], [12] — provide the most abstract approaches to requirement specification and real-time system analysis [13], [1].

Among temporal logics an important distinction must be made in order to identify their expressiveness. Many temporal logics are based on propositional logic — e.g., PTL [3], [14], TPTL [15], RTTL [16], ITL [17] — and adopt the $\Diamond$ and $\Box$ operators. Other temporal logics defined more recently are based on first or higher order logic — e.g., TRIO [18], MTL [19], [20], interval temporal logic [21]. Propositional logic is decidable, while FOL has a greater expressive power but it is intrinsically undecidable; however, some restrictions can be applied to make the theory both decidable and executable [17], [22]. Higher order logics have an even greater expressive power, but are more difficult to manipulate automatically than simpler logics. For these reasons, the most expressive temporal logics are based on FOL and, if executability of the specification is required, restrictions are applied. Consequently, most of the temporal logics can be translated into FOL. Their definition is very useful since temporal logics constrain the user to write formulæ whose validity and satisfiability can be more easily checked, leading to specifications that can be verified or automatically validated.

Logic-based languages capable of modeling temporal constraints can be divided into two main categories – those based on *time points* or those based on *time intervals* – depending on the temporal semantics adopted. For logics based on *time points* [23], [24], temporal expressions 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. In the case of *time intervals* IL [25], [5], ITL [17], [26], [27], interval temporal logic in [21], [28], EIL [29], RTIL [30] GIL [31], formulæ specify the temporal relationships among facts, events, and intervals, thus allowing a higher level of abstraction. These logics usually have specific operators to express the relationships between intervals (*meet*, *before*, *after* [32]), operators for combining intervals (e.g., the *chop* operator [24]), or operators that specify the interval constituting the context of temporal formulæ [29]. Interval logics are typically more concise that point-based temporal logics since the specification of temporal constraints on intervals occurs frequently in real-time systems.

The time structure of a logical language can be linear or branched; however, only a linear structure can be productively used for system specification. A branched future can be unsuitable for specification languages, since for branched models the system can have more than one possible evolution and the correct evolution cannot be unequivocally determined in advance (for a more detailed comparison please consult [33], [34]).

The time domain can be modeled as either a discrete or as a continuous domain by using natural or real numbers, respectively. In the discrete case, the specification can be regarded as a set of possible states where the synchronization among events can be deduced from the specification, since these events are placed on a regular time grid. In the continuous case, the specification has an unpredictable number of states and the synchronization among events must be explicitly declared, since each event can be distant from another event by an infinitesimal amount.

The problem of executability of specifications (in temporal logics) has often been misunderstood, mainly because there are at least three different definitions of executability [35]. In many cases, specification models are considered 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. In effect, this property corresponds more to the decidability of the validity problem rather than to executability. Another definition of executability refers to the possibility of generating a model for a given specification (i.e., a history of input/output values) [36]. The third definition of executability refers to using the specification itself as a prototype of the real-time system, thus allowing, in each time instant, the *on-line* generation of system outputs on the basis of present inputs, its internal state, and its past history. When this is possible, the specification can be executed for testing, just as in operational approaches. With respect to this last definition, many examples of logics that are supposed to have executable semantics, but in reality cannot be used to build an executable prototype [35], exist in t he literature. Therefore, dual models have been recently proposed [37], [16], [38], [39]. The dual models try to integrate operational and descriptive capabilities to allow both the mathematical proof of properties and the executability of system specification.

None of the temporal logics presented in the past few years is completely satisfactory for real-time system specification. In fact, most of them have no metric for time, thus allowing only specification of qualitative temporal requirements — e.g., [3], [14], [5]. In the literature, only a few examples of quantitative temporal logics exist. In these cases, an operator expressing the distance between time points is usually defined. Most of the approaches including the metric for time are based on propositional logic instead of FOL, and are therefore not expressive enough to describe realistic systems - e.g., [40], EIL [29], RTIL [30], TPTL [15], [22]. Those first-order temporal logics that provide a metric for time usually allow quantification over the temporal domain — e.g., RTL [4], MTL [20], TRIO [18] — whereas a prohibition of this kind of quantification has been shown to be a necessary condition for the existence of feasible automated verification mechanisms [41]. All these temporal logics are based on time points rather than on intervals, and provide a sharp distinction between past and future.

The authors have concentrated on most of the above-mentioned factors in order to define a powerful temporal logic, with special emphasis on its expressiveness and conciseness. This Temporal Interval Logic with Compositional Operators (TILCO) is based on time intervals. TILCO has been especially designed for the specification of real-time systems. TILCO extends FOL with a set of temporal operators and can be regarded as a generalization of the classical temporal logics based on the application of the operators *eventually* and *henceforth* to time intervals.

TILCO has a metric for time, the time is discrete and no explicit temporal quantification is allowed. Thus, TILCO allows specification of both qualitative and quantitative relationships about events and facts and provides specific compositional operators among time intervals. 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 bases of the specification in the system validation phase. 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. In fact, TILCO can be effectively used to express invariants, precedence among events, periodicity, liveness and safety conditions, etc., and these properties can be formally verified by automatic theorem-proving techniques. To this end, a formalization of TILCO has been implemented in the theorem prover Isabelle/HOL [42], [43]. 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 [1].

This paper is organized as follows. Section 2 presents TILCO's syntax and semantics. Section 3 discusses the axiomatization of TILCO theory and its soundness and decidability. Section 4 provides the mechanisms for system validation by means of high-level properties proof and a brief overview of the executor of TILCO specifications. Section 5 compares TILCO to others first order temporal logics in terms of conciseness and expressiveness. Section 6 provides a complete example of specification to show the language capabilities. Conclusions are drawn in Section 7.

2 Definition of TILCO

This section provides details about the syntax and semantics of TILCO. TILCO extends FOL in order to create a logic language capable of specifying the relationships between events and time, as well as the transformations on the data domain. It can be used to specify temporal constraints among events in either a qualitative or quantitative manner. Therefore, the boundaries of an interval, which specify the length of intervals and actions, can be expressed relative to other events (i.e., in a qualitative manner) or with an absolute measure (i.e., in a quantitative manner). This allows definition of expressions of ordering relationships among events or delays and time-outs. These features are mandatory for specifying the behavior of real-time systems. In addition, the TILCO deductive approach is sound, and thus consistent. It forces the user to write formulæ without using direct quantifications over the temporal domain, thus avoiding the writing overly intricate or difficult to understand specifications [22].

TILCO includes the concepts of typed variables and constants; it provides a set of basic types and allows the definition of new types by means of the mechanisms of enumerated collection and type constructors (see App.A). A type-checking mechanism is automatically extended to these new types. The predefined types are: **nat** for natural numbers, **int** for integer numbers, **bool** for Booleans, **char** for text characters, and **string** for character strings. The usual arithmetic operators: $+$, $-$, $*$, $/$, mod, $\sim$ (change sign), are defined for integers and natural numbers. String manipulation functions are defined for strings. Comparative operators: $=$, $<$, $>$, $\geq$, $\leq$, $\neq$, can be used with integers, naturals, characters and strings, and they can also be overloaded for dealing with user-defined types.

A system specification in TILCO is a tuple

$$\{\mathcal{U}, \mathcal{T}, \mathcal{F}, \mathcal{P}, \mathcal{V}, \mathcal{W}, \mathcal{C}, \mathcal{J}\},$$

where $\mathcal{U}$ is a set of TILCO formulæ, $\mathcal{T}$ a set of type definitions, $\mathcal{F}$ a set of functions, $\mathcal{P}$ a set of predicates, $\mathcal{V}$ a set of typed time-dependent variables, $\mathcal{W}$ a set of typed time-independent variables, $\mathcal{C}$ a set of typed constants (also called time invariant parameters), and $\mathcal{J}$ is a set of integer intervals. $\mathcal{U}$ specifies the rules defining the behavior of the specified system. $\mathcal{T}$ defines the types used in the specification. Functions and predicates have their usual meaning and are used to manipulate predefined and user-defined data-types. Time-dependent variables are employed for modeling system inputs (read-only), outputs (write-only), and auxiliary variables (read/write) of the system under specification. Time-dependent variables can assume any value in their corresponding domain. Time-independent variables are used to build parametric formulæ that operate on structured data types (i.e., arrays, lists, etc.) through quantification. Constants are used for modeling system parameters. Integer intervals, which are connected sets of integers, are used for specifying quantitative temporal relationships.

A system is specified in TILCO according to the following rules:

- a system is characterized by its input and output ports, which are used to communicate with the external environment, and by its auxiliary variables, defining its internal state;
- inputs, outputs and auxiliary variables can assume only one value at each time instant. Each of them is defined by a unique name;
- an input is a typed variable whose value can change due to external events;
- an output is a typed variable which can be forced to assume a value by some predicates through an assignment. This leads to a change in the external environment;
- an auxiliary variable can be forced to a value by an assignment and it can be read as an input variable;
- a system is described to be a set of formulæ which define its behavior and the data transformation.

2.1 Syntax and semantics of TILCO

TILCO's *temporal operators* have been added to FOL by leaving the evaluation time implicit. Therefore, the meaning of a TILCO formula is given with respect to the current time such as in other logic languages — e.g., [18], [44]. Time is discrete and linear, and the temporal domain is $\mathcal{Z}$, the set of integers; the minimum time interval corresponds to 1 time unit. The current time instant is represented by 0, whereas positive (negative) number represent future (past) time instants. TILCO formulæ can be time dependent or independent; the latter are those that do not present any TILCO *temporal operator*, and are comprised only of time-independent subformulæ. A time independent formula can be regarded as a constraint that must be satisfied in each time instant.

The basic temporal entity in TILCO is the interval. Intervals can be quantitatively expressed by using the notation with round, "(", ")", or squared, "[", "]", brackets for excluding and including interval boundaries, respectively. Time instants are regarded as special cases that are represented as closed intervals composed of a single point (e.g., $[a, a]$). Symbols $+\infty$ and $-\infty$ can be used as interval boundaries, if the extreme is open, to denote infinite intervals — i.e., $[a, +\infty)$ represents set $\{x \in \mathcal{Z} | a \leq x\}$. In this way, TILCO allows both the specification of *facts* in intervals and *events* in time instants. Classical operators of temporal logic (i.e., eventually, $\diamond$, and henceforth, $\Box$) can be easily obtained by using TILCO operators with infinite intervals. For these reasons, TILCO can be regarded as a generalization of most of the interval logics presented in the literature in the past — e.g., [5], [4], [3] — with the addition of a metric to measure time.

The basic TILCO *temporal operators* are:

- "**@**", bounded universal temporal quantification over an interval;
- "**?**", bounded existential temporal quantification over an interval;
- **until**, to express that either a predicate will always be true in the future, or it will be true until another predicate will become true;
- **since**, to express that either a predicate has always been true in the past, or it has been true since another predicate has become true.

Operators "**@**" and "**?**" are called temporal quantifiers. $A@i$ is true if formula A is true in every instant in the interval i, with respect to the current time instant. Therefore, if t is the current time instant, $(A@i)^{(t)} \equiv \forall x \in i.A^{(x+t)}$ holds. In particular, $A@[t_1, t_2)$ evaluated in t means:

$$\forall x \in [t_1, t_2).A^{(x+t)}.$$

Obviously t_1 and t_2 can be either positive or negative, and, thus the interval can be in the past or in the future. If the lower bound of an interval is greater than the upper bound, then the interval is null (i.e., it is equal to the empty set). Operators "**@**" and "**?**" correspond, in the temporal domain, to FOL quantifiers $\forall$ and $\exists$, respectively; hence, they are related by a duality relationship analogous to that between $\forall$ and $\exists$. "@" and "?" are used to express delays, time-outs and any other temporal constraint that requires a specific quantitative bound. Concerning the other *temporal operators*, **until** A B (evaluated in t) is true if B will always be true in the future with respect to t, or if B will be true in the interval $(t, x+t)$ with $x > 0$ and A will be true in $x+t$. This definition of **until** does not require the occurrence of A in the future, so the **until** operator corresponds to the *weak until* operator defined in PTL [14]. The operators **until** and **since** express the same concept for future and past, respectively; they are related by a relationship of *temporal duality*. **until** and **since** can be effectively used to express ordering relationships among events without the need of specifying any numeric constraint.

Given $\mathcal{F}$, $\mathcal{P}$, $\mathcal{V}$, $\mathcal{W}$, $\mathcal{C}$, $\mathcal{J}$, the syntax of TILCO formulæ is defined by the following BNF-like definitions:

$$
\begin{array}{rcl}
\text{interval} & ::= & (a,b)|(a,b]|[a,b)|[a,b] \quad \textit{for each } a,b \in \mathcal{Z} \\
\text{interval\_list} & ::= & \text{interval} \\
 & | & \text{interval interval\_op interval} \\
\text{interval\_op} & ::= & ,\,|; \\
\text{variable} & ::= & w \quad \textit{for each } w \in \mathcal{W} \\
\text{term} & ::= & v \quad \textit{for each } v \in \mathcal{V} \\
 & | & \text{variable} \\
 & | & c \quad \textit{for each } c \in \mathcal{C} \\
 & | & f(\text{term\_list}) \quad \textit{for each} f \in \mathcal{F} \\
\text{term\_list} & ::= & \text{term} \\
 & | & \text{term, term\_list} \\
\text{atomic\_formula} & ::= & p(\text{term\_list}) \quad \textit{for each } p \in \mathcal{P} \\
\text{formula} & ::= & \top|\bot|\text{atomic\_formula} \\
 & | & \neg\text{formula} \\
 & | & \text{formula op formula} \\
 & | & v := \text{term} \quad \textit{for each } v \in \mathcal{V} \\
 & | & \text{quantifier variable. formula} \\
 & | & \text{formula temporal\_quantifier interval\_list} \\
 & | & \text{temporal\_op formula formula} \\
 & | & (\text{formula}) \\
\text{op} & ::= & \vee|\wedge|\Rightarrow|\Leftrightarrow|=\!\rhd\!\rhd|=\!\lhd\!\lhd \\
\text{quantifier} & ::= & \forall|\exists|\exists! \\
\text{temporal\_quantifier} & ::= & @|? \\
\text{temporal\_op} & ::= & \textbf{until}|\textbf{since}
\end{array}
$$

The use of parentheses in TILCO expressions is reduced by using the operators' precedence relationships reported in Tab. 1.

prec.	operators
1	$\sim$
2	$*$ $/$ mod
3	$+$ $-$
4	$=$ $>$ $<$ $\geq$ $\leq$ $\neq$
5	$\neg$ $,$ $;$
6	$:=$
7	@ **?**
8	$\wedge$
9	$\vee$
10	$\Leftrightarrow$ $\Rightarrow$ $=\!\rhd\!\rhd$ $=\!\lhd\!\lhd$
11	$\forall$ $\exists$ $\exists!$
12	**until** **since**

Table 1: Precedences among TILCO operators.

Before defining the semantics of TILCO, it is important to introduce the concept of *interpretation* of a TILCO formula. This concept is also used to define the validity and the satisfiability of TILCO formulæ.

Given a syntactically correct TILCO formula A, with $\{t_1, \ldots, t_h\}$ set of types used in A, $\{p_1, \ldots, p_k\}$ predicates, $\{f_1, \ldots, f_l\}$ functions, $\{v_1, \ldots, v_m\}$ time-dependent variables, $\{c_1, \ldots, c_q\}$ constants, and $\{j_1, \ldots, j_r\}$ intervals present in A, then an *interpretation* $\mathcal{I}$ is a tuple

$$(\{D_1, \ldots, D_h\}, \{R_1, \ldots, R_k\}, \{F_1, \ldots, F_l\}, \{V_1(t), \ldots, V_m(t)\}, \{C_1, \ldots, C_q\}, \{J_1, \ldots, J_r\})$$

where:

- $\{D_1, \ldots, D_h\}$ assigns a domain D_i to each type t_i;
- $\{R_1, \ldots, R_k\}$ assigns an n-ary relation R_i over $D_{i_1} \times \ldots \times D_{i_n}$ to each n-ary predicate p_i with arguments of type $t_{i_1}, \ldots, t_{i_n}$;
- $\{F_1, \ldots, F_l\}$ assigns an n-ary function F_i over $D_{i_1} \times \ldots \times D_{i_n}$ to each n-ary function f_i with arguments of type $t_{i_1}, \ldots, t_{i_n}$;
- $\{V_1(t), \ldots, V_m(t)\}$ assigns a function of time $V_i(t) : \mathcal{Z} \rightarrow D_n$ to each time-dependent variable v_i of type t_n, specifying the history of that variable in every time instant (where t is the absolute time);
- $\{C_1, \ldots, C_q\}$ assigns a value $C_i \in D_n$ to each constant c_i of type t_n;

- $\{J_1, \ldots, J_r\}$ assigns an interval value J_i to each integer interval j_i.

Given a TILCO formula A and an interpretation $\mathcal{I}$ for A, notation

$$\mathcal{I}, t \models A$$

expresses that $\mathcal{I}$ is a *model* for A evaluated in the time instant t. The evaluation of $\mathcal{I}, t \models A$, stating the semantics of TILCO, is inductively defined on the structure of A by the following rules:

- $\mathcal{I}, t \models \top$;
- $\mathcal{I}, t \not\models \bot$;
- $\mathcal{I}, t \models \neg A$ iff $\mathcal{I}, t \not\models A$;
- $\mathcal{I}, t \models A_1 \wedge A_2$ iff $\mathcal{I}, t \models A_1$ and $\mathcal{I}, t \models A_2$;
- $\mathcal{I}, t \models A_1 \vee A_2$ iff either $\mathcal{I}, t \models A_1$ or $\mathcal{I}, t \models A_2$;
- $\mathcal{I}, t \models A_1 \Rightarrow A_2$ iff $\mathcal{I}, t \models \neg A_1 \vee A_2$;
- $\mathcal{I}, t \models A_1 {=}\!\rhd\!\!\rhd A_2$ iff either $\mathcal{I}, t \models \neg A_1$ or $\mathcal{I}, t+1 \models A_2$;
- $\mathcal{I}, t \models A_1 {=}\!\lhd\!\!\lhd A_2$ iff either $\mathcal{I}, t \models \neg A_1$ or $\mathcal{I}, t-1 \models A_2$;
- $\mathcal{I}, t \models A_1 \Leftrightarrow A_2$ iff $\mathcal{I}, t \models A_1 \Rightarrow A_2 \wedge A_2 \Rightarrow A_1$;
- $\mathcal{I}, t \models x := exp$ iff there exists a constant $k \in D_x$ such that $\mathcal{I}, t \models x = k$ and $\mathcal{I}, t-1 \models exp = k$, where D_x is the domain assigned to the type of x by $\mathcal{I}$;
- $\mathcal{I}, t \models \forall x.A(x)$ iff, for each $y \in D_x$ it is true that $\mathcal{I}, t \models A(y)$, where D_x is the domain assigned to the type of x by $\mathcal{I}$;
- $\mathcal{I}, t \models \exists x.A(x)$ iff, there exists a $y \in D_x$ such that $\mathcal{I}, t \models A(y)$, where D_x is the domain assigned to the type of x by $\mathcal{I}$;
- $\mathcal{I}, t \models \exists! x.A(x)$ iff, there exists one and only one $y \in D_x$ such that $\mathcal{I}, t \models A(y)$, where D_x is the domain assigned to the type of x by $\mathcal{I}$;
- $\mathcal{I}, t \models A@i$ iff, for each $s \in i$, $\mathcal{I}, s+t \models A$ is true;
- $\mathcal{I}, t \models A?i$ iff, there exists an $s \in i$ such that $\mathcal{I}, s+t \models A$;
- $\mathcal{I}, t \models$ **until** A_1 A_2 if either $\mathcal{I}, t \models A_2@(0, +\infty)$ or there exists $\tau > 0$ such that $\mathcal{I}, t+\tau \models A_1$ and $\mathcal{I}, t \models A_2@(0, \tau)$;
- $\mathcal{I}, t \models$ **since** A_1 A_2 if either $\mathcal{I}, t \models A_2@(-\infty, 0)$ or there exists $\tau < 0$ such that $\mathcal{I}, t+\tau \models A_1$ and $\mathcal{I}, t \models A_2@(\tau, 0)$;

- $\mathcal{I}, t \models A@i, j$ iff $\mathcal{I}, t \models (A@i) \wedge (A@j)$;
- $\mathcal{I}, t \models A?i, j$ iff $\mathcal{I}, t \models (A?i) \wedge (A?j)$;
- $\mathcal{I}, t \models A@i; j$ iff $\mathcal{I}, t \models (A@i) \vee (A@j)$;
- $\mathcal{I}, t \models A?i; j$ iff $\mathcal{I}, t \models (A?i) \vee (A?j)$;
- $\mathcal{I}, t \models p_i(e_1, \ldots, e_n)$, iff $(E_1, \ldots, E_n) \in R_i$, where R_i is the relation assigned by $\mathcal{I}$ to p_i and E_j, for each $j = 1, \ldots, n$, are the results of the expressions e_j when the values assigned by $\mathcal{I}$ are substituted for the constants and variables, and the variables are evaluated in t.

The semantics of predicates also includes that of functions, variables and constants.
Remark 2.1 In the case where the interval is null, it holds:

$$\begin{array}{rcl} A@\emptyset & = & \top; \\ A?\emptyset & = & \bot. \end{array}$$

□
Some useful definitions follow.

Definition 2.1 *Given an interpretation*

$$\mathcal{I} = \left\{ \begin{array}{l} \{D_1, \ldots, D_h\} \\ \{R_1, \ldots, R_k\} \\ \{F_1, \ldots, F_l\} \\ \{V_1(t), \ldots, V_m(t)\} \\ \{C_1, \ldots, C_q\} \\ \{J_1, \ldots, J_r\} \end{array} \right\},$$

its temporal translation by $s \in \mathcal{Z}$ time units is defined by:

$$\tau(\mathcal{I}, s) \stackrel{\text{def}}{=} \left\{ \begin{array}{l} \{D_1, \ldots, D_h\} \\ \{R_1, \ldots, R_k\} \\ \{F_1, \ldots, F_l\} \\ \{V_1(t+s), \ldots, V_m(t+s)\} \\ \{C_1, \ldots, C_r\} \\ \{J_1, \ldots, J_r\} \end{array} \right\}.$$

Definition 2.2 *A TILCO formula A is said to be* satisfiable *if there exists an interpretation $\mathcal{I}$ and a value $t \in \mathcal{Z}$ such that $\mathcal{I}, t \models A$.*

Definition 2.3 *A TILCO formula A is said to be* valid in an interpretation $\mathcal{I}$ *if for each $t \in \mathcal{Z}$ it is true that $\mathcal{I}, t \models A$. The notation used is $\mathcal{I} \models A$.*

Definition 2.4 *A TILCO formula A is said to be* valid *if for each interpretation $\mathcal{I}$ and for each $t \in \mathcal{Z}$ it is true that $\mathcal{I}, t \models A$. The notation used is $\models A$.*

Definition 2.5 *Given a set of TILCO formulæ, $U = \{A_1, \ldots, A_n\}$, U is said to be* satisfiable *if there exists an interpretation $\mathcal{I}$ such that $\mathcal{I} \models A_1, \ldots \mathcal{I} \models A_n$. $\mathcal{I}$ is said to be a* model *for U. U is said to be* unsatisfiable *if for each $\mathcal{I}$ there exists an i such that $\mathcal{I} \not\models A_i$.*

Definition 2.6 *Given a set of TILCO formulæ U and a TILCO formula A, if every model $\mathcal{I}$ for U is such that $\mathcal{I} \models A$, then A is said to be a* logic consequence *of U. The notation used is $U \models A$.*

Definition 2.7 *Given $\mathcal{S}(U) = \{A | U \models A\}$, $\mathcal{S}(U)$ is called* theory *of U and the elements of $\mathcal{S}(U)$ are called* theorems *of U. The elements of U are called* axioms *of $\mathcal{S}(U)$.*

2.2 Comments

- In a TILCO specification, a system is described by a formula consisting of the conjunction of all the formulæ of $\mathcal{U}$, each describing a different aspect of the system. A specification is defined in a *specification temporal domain* by means of operator "@". For example, if $\mathcal{U} = \{F_1, F_2, F_3\}$ and the temporal domain is i, then the system is described by:

$$(F_1 \wedge F_2 \wedge F_3)@i,$$

  which means that all properties F_1, F_2, F_3 must be valid in each time instant of i.

- Each TILCO formula used in a system specification must be closed, in the sense that each time independent variable in a formula must be quantified. For instance, formula $\exists s.f(k, s) \Rightarrow P$ is open, while $\exists s.\exists k.f(k, s) \Rightarrow P$ is closed. If a TILCO formula is open, it is replaced by its universal closure (i.e., an external universal quantifier is introduced for each of the time independent variables which are not quantified). According to the syntax definition, each quantified variable must be time independent, otherwise (i) it would be possible to write higher order formulæ and (ii) time could not be left implicit because the meaning of the formula would change during system evolution.

- In a TILCO specification, predicates and functions with typed parameters can also be defined. Predicates are functions that return a value of type **bool**. Functions and predicates are used to define operations and relationships over predefined and user-defined types. Functions and predicates are incrementally defined by using predefined functions and predicates over the basic data types and type constructors. The body of each predicate must be specified by means of a TILCO formula, in which the only non-quantified variables are the predicate parameters. Predicates are only instruments used to simplify the writing of formulæ; hence, more complex temporal expressions and formulæ can be hidden in predicates. These also extend the number of temporal operators of TILCO, since they can be used to constitute a user-defined library of predicates, thus improving the specifications reusability. For example, a predicate for specifying that A occurs only once in an interval i could be defined as:

$$\text{OnlyOnce}(A : \text{int} \rightarrow \text{bool}, i : \text{interval}) : \text{bool} \stackrel{\text{def}}{=} \exists! m.A(m)?i,$$

where each occurrence of A is characterized by a different value of m:

$$\models \forall m.A(m) \Rightarrow \neg(A(m)?(-\infty, 0)),$$

so that $\exists! m.A(m)?i$ specifies that the event A happens only once during the interval i. m can be regarded as a *time-stamp*. The adoption of *time-stamps* for distinguishing different occurrences of events has been introduced in [19], in order to overcome the limitation of temporal logics in recognizing different occurrences of an event. Since TILCO is an extension of FOL, the use of *time-stamps* in specifications is simply obtained by adding them to predicates whose different occurrences must be distinguished.

- The two predicates

$$\begin{aligned} \text{rule}(A : \text{bool}) &\stackrel{\text{def}}{=} A@(-\infty, +\infty), \\ \text{fact}(A : \text{bool}) &\stackrel{\text{def}}{=} A?(-\infty, +\infty), \end{aligned}$$

  express that a predicate A is always or sometimes true, respectively. These predicates are often used in specifications to express the concepts of necessity and possibility over the whole temporal domain.

- The classical *henceforth* operator, $\Box$, can be expressed in terms of TILCO operator "@": $A@[0, +\infty)$, which means that A will be true forever from the current time instant. Analogously, the *eventually* operator, $\Diamond$, can be expressed by $A?[0, +\infty)$.

- Operator "?" could also be defined in terms of operator "@" by using the duality relationship:

$$A?i \equiv \neg(\neg A@i).$$

- In order to simplify writing specifications the symbol $=\!\rhd\!\rhd$ ($=\!\lhd\!\lhd$) has been introduced to express that a formula implies that another formula will be (has been) true at the next (previous) time instant:

$$\begin{aligned} A =\!\rhd\!\rhd B &\equiv A \Rightarrow B@[1, 1], \\ A =\!\lhd\!\lhd B &\equiv A \Rightarrow B@[-1, -1]. \end{aligned}$$

- 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 "?". Other operators among intervals, such as intersection, "$\cap$", and union, "$\cup$", could be defined by considering time intervals as sets. However, the introduction of $\cup$ is problematic because the set of intervals is not closed over this operation.

$A@[1,1]$	A will be true at the next time instant
$A@[0,t)$	A is true from now for t time instants
$A@(-\infty,+\infty)$	A has been, is and will be always true
$A@(0,+\infty)$	A will be always true in the future
$A?(0,+\infty)$	A will be sometimes true in the future
$A@[t_1,t_2]$	A is true in $[t_1,t_2]$
$A?[t_1,t_2)$	A is true in an instant of $[t_1,t_2)$
$\neg(A@(-\infty,+\infty))$	A is not always true
$\neg A@(-\infty,+\infty)$	A is always false
$A@[t_1,t_1],(t_2,t_3]$	A is true in t_1, and in $(t_2,t_3]$
$A?[t_1,t_1],(t_2,t_3]$	A is true in t_1, and is true at least once in $(t_2,t_3]$
$A@[t_1,t_1];(t_2,t_3]$	A is true in t_1, or in $(t_2,t_3]$
$A@[t,t]\wedge\neg A@(0,t)$	t is the next time instant in which A will be true
$A@[-t,-t]\wedge\neg A@(-t,0)$	$-t$ is the last time instant in which A has been true
$A?[0,t_1]@[0,+\infty)$	A will become true within t_1 for each time instant in the future (response)
$A@[0,t_1]?[0,+\infty)$	A will be true, and since then it will remain true for t_1 time units (persistence)
$(A\Rightarrow B)@[0,+\infty)$	A causes B always in the future
$(A\Rightarrow B)?[0,t]$	if A is true within t, then also B will be true at the same time
$(A\Rightarrow B?i)@j$	A leads to an assertion of B in i for each time instant of j
$(A\Rightarrow B@i)@j$	A leads to the assertion of B in the whole interval i for each time instant of j
$(A\Rightarrow B@i)?j$	A leads to the assertion of B in the whole interval i in at least a time instant of j

Table 2: Examples of TILCO formulæ.

2.3 Short examples

Tab. 2 provides examples of TILCO formulæ. To provide a clearer view of TILCO's expressiveness the formulæ are accompanied by an explanation of their meaning. In Tab. 2, t stands for a positive integer number.

A bit more complex example is a formula that specifies a system with an input I_1 : **int** and an output O_1 : **bool**. The system produces an output signal for t_1 time instants with a delay of t_2 time instants every time that the input assumes the value *val*:

$$I_1 = val \Rightarrow O_1@[t_2, t_1 + t_2].$$

The same system is also specified by the formula:

$$I_1 = val \Rightarrow O_1@[0, t_1]@[t_2, t_2].$$

Another example is the specification of a system for generating periodic events:

$$(\neg B@(0,10] \Leftrightarrow (B@(10,20] \wedge \neg B@(20,30])) \wedge ((\neg B?[-1,-1] \wedge B) \Leftrightarrow (A@[0,2] \wedge \neg A@(2,20))).$$

This TILCO formula specifies that signal B is periodic with a duty-cycle of 50 percent and a period of 20 time units while, being associated with each transition of B (from false to true) signal A stays true for 2 time units. Fig. 1 depicts the histories of signals A and B.

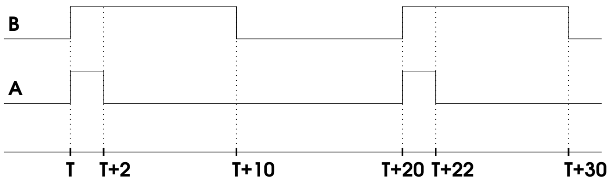

Figure 1: Histories of signals A and B.

Once system behavior is specified by means of a set of TILCO formulæ, the specification can be validated to verify whether it corresponds to the system requirements. In TILCO, system validation is performed by proving that high-level properties (e.g., safety, liveness, etc.) are satisfied by the TILCO specification of the system. These properties can be expressed by means of other TILCO formulæ, thus TILCO is used to specify both the system and its high-level properties. Therefore, the classical safety conditions, such as $A@i$ (where A is a positive property), and $\neg B@i$ where B is a negative condition) 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. Moreover, liveness conditions, such as $A?i$ (A will be satisfied within i) or deadlock-free conditions, such as $(\neg A?i)@j$, can also be specified. If during the validation of a TILCO specification it is found that a desired property (constituting a system requirement) cannot be deduced from the system specification given in terms of TILCO formulæ, then the specification is incomplete. If that property must be satisfied by the system, a new TILCO formula should be added to the system specification, provided that this formula does not contradict any other formula contained in the specification. This formula may itself be the desired property or a formula that completes the system specification in order to prove the desired property, thus allowing the incremental system specification.

3 Axiomatization

This section presents an axiomatization for TILCO, extending the Hilbert axiom system for FOL. Axioms and inference rules dealing with TILCO *temporal operators* have been added to build a sound deduction system for TILCO. In particular, four new axioms have been added to deal with the operators **until** and **since**, as well as two new inference rules for operator "@". Since TILCO is an extension of FOL, the classical properties of FOL are valid theorems in TILCO. In fact, all these properties can still be deduced by using the FOL axioms contained in the TILCO axiomatization. The axiomatization for TILCO is divided in three parts: (i) the axioms for FOL, (ii) the axioms for TILCO *temporal operators*, and (iii) the inference rules. In stating axioms and rules

- $\vdash A$, means that A is provable in every time instant;
- $\vdash_t A$, means that A is provable in the time instant t.

These two notations are related by the following rules which have been proven:

$$TG \quad \frac{\vdash_t A}{\vdash A} \quad \text{provided that } t \text{ is not free in any assumption,}$$

$$TS \quad \frac{\vdash A}{\vdash_t A},$$

which are called *temporal generalization* (TG) and *temporal specialization* (TS), respectively.

First order axioms

The Hilbert axioms for FOL are [45], [14]:

AX1 $\vdash B \Rightarrow (A \Rightarrow B)$;

AX2 $\vdash (A \Rightarrow (C \Rightarrow D)) \Rightarrow ((A \Rightarrow C) \Rightarrow (A \Rightarrow D))$;

AX3 $\vdash (\neg A \Rightarrow \neg B) \Rightarrow ((\neg A \Rightarrow B) \Rightarrow A)$;

AX4 $\vdash (\forall x.A(x)) \Rightarrow A(a)$, if a is a free term for x;

AX5 $\vdash (\forall x.A \Rightarrow B(x)) \Rightarrow (A \Rightarrow (\forall x.B(x)))$, if A does not depend on x.

TILCO axioms

The following axioms describe the essential properties of TILCO operators **until** and **since**:

AX6 $\vdash$ **until** $A\ B \Leftrightarrow A@[1,1] \vee ((B \wedge \textbf{until}\ A\ B)@[1,1])$;

AX7 $\vdash$ **since** $A\ B \Leftrightarrow A@[-1,-1] \vee ((B \wedge \textbf{since}\ A\ B)@[-1,-1])$;

AX8 $\vdash B@(0,+\infty) \Rightarrow$ **until** $A\ B$;

AX9 $\vdash B@(-\infty,0) \Rightarrow$ **since** $A\ B$.

Axioms 6 and 7 are used for induction setup in order to prove propositions containing operators **until** and **since**, respectively. Axioms 8 and 9 constitute the basic cases for induction over **until** and **since**, respectively.

Inference rules

The most important inference rule adopted in the TILCO theory is the Modus Ponens (MP):

$$\frac{\vdash A \quad \vdash A \Rightarrow B}{\vdash B},$$

with its classical associated meaning. Moreover, the Generalization (GEN) rule is adopted for quantifier $\forall$:

$$\frac{\vdash A(a)}{\vdash \forall x.A(x)},$$

where A can either depend or not depend on x, and x must be a time independent variable — i.e., a variable that can be used to quantify over the elements of a set. With the above inference rules, the first order theory underlying TILCO can be regarded as a standard theory with respect to time-independent parameters and predicates.

In order to deal with operator "@", two new deduction rules in natural deduction style were introduced instead of adding new axioms. These two rules, $@E$ and $@I$, allow the decomposition and the composition of formulæ containing "@", respectively:

$$@E \quad \frac{\vdash_t A@i \quad \vdash x \in i}{\vdash_{t+x} A},$$

$$@I \quad \frac{x \in i \vdash_{t+x} A}{\vdash_t A@i} \quad \text{provided that } x \text{ is not free in any assumption.}$$

Other general theorems

FOL's deductive system in natural deduction style has been enhanced by adding rules for introduction and eliminating TILCO *temporal operators*:

$$\top I \quad \frac{}{\vdash \top} \qquad \bot E \quad \frac{\vdash \bot}{\vdash P}$$

$$\wedge I \quad \frac{\vdash P \quad \vdash Q}{\vdash P \wedge Q} \qquad \wedge E1 \quad \frac{\vdash P \wedge Q}{\vdash P} \qquad \wedge E2 \quad \frac{\vdash P \wedge Q}{\vdash Q}$$

$$\vee E \quad \frac{\vdash P \vee Q \quad P \vdash R \quad Q \vdash R}{\vdash R} \qquad \vee I1 \quad \frac{\vdash P}{\vdash P \vee Q} \qquad \vee I2 \quad \frac{\vdash Q}{\vdash P \vee Q}$$

$$\Rightarrow I \quad \frac{P \vdash Q}{\vdash P \Rightarrow Q} \qquad \Rightarrow E(\mathrm{MP}) \quad \frac{\vdash P \Rightarrow Q \quad \vdash P}{\vdash Q}$$

$$\forall I \quad \frac{\vdash P(x)}{\vdash \forall x.P(x)} \ x \text{ free only in } P \qquad \forall E \quad \frac{\vdash \forall x.P(x)}{\vdash P[t/x]}$$

$$\exists I \quad \frac{\vdash P[t/x]}{\vdash \exists x.P(x)} \qquad \exists E \quad \frac{\vdash \exists x.P(x) \quad P(x) \vdash Q}{\vdash Q} \ x \text{ free only in } P$$

Rule $\Rightarrow I$ is also called *deduction rule* (DR), rule $\forall I$ is also called *generalization rule* (GEN), and $\exists E$ is called *existential instantiation rule* (EI). An alternative way of stating rule EI is:

$$\frac{\vdash \exists x.P(x)}{\vdash P(a)} \quad \text{where } a \text{ is a new constant}$$

that is sometimes simpler to use.

The following introduction and elimination rules have been specifically proven for TILCO operators:

$$:= I \quad \frac{\vdash_t v = k \quad \vdash_{t-1} e = k}{\vdash_t v := e} \ k \text{ constant} \qquad := E \quad \frac{\vdash_t v := e \quad \vdash_{t-1} e = k}{\vdash_t v = k} \ k \text{ constant}$$

$$@I \quad \frac{x \in i \vdash_{x+t} P}{\vdash_t P@i} \ x \text{ not free in any assump.} \qquad @E \quad \frac{\vdash_t P@i \quad \vdash x \in i}{\vdash_{x+t} P}$$

$$?I \quad \frac{\vdash_{x+t} P \quad \vdash x \in i}{\vdash_t P?i} \qquad ?E \quad \frac{\vdash_t P?i \quad \frac{\vdash_{x+t} P \quad \vdash x \in i}{\vdash R}}{\vdash R} \ x \text{ not free in any assump.}$$

$$\mathrm{until}I1 \quad \frac{\vdash_{t+x} P \quad \vdash_t Q@(0,x) \quad \vdash 0 < x}{\vdash_t \text{until } P\ Q} \qquad \mathrm{until}I2 \quad \frac{\vdash_t Q@(0,+\infty)}{\vdash_t \text{until } P\ Q}$$

$$\mathrm{since}I1 \quad \frac{\vdash_{t+x} P \quad \vdash_t Q@(x,0) \quad \vdash x < 0}{\vdash_t \text{since } P\ Q} \qquad \mathrm{since}I2 \quad \frac{\vdash_t Q@(-\infty,0)}{\vdash_t \text{since } P\ Q}$$

$$\text{until}E \quad \frac{\vdash_t \text{until } P\ Q \quad \frac{\vdash_{t+x} P \quad \vdash_t Q@(0,x) \quad \vdash 0<x}{\vdash R} \quad \frac{\vdash_t Q@(0,+\infty)}{\vdash R}}{\vdash R}$$

$$\text{since}E \quad \frac{\vdash_t \text{since } P\ Q \quad \frac{\vdash_{t+x} P \quad \vdash_t Q@(x,0) \quad \vdash x<0}{\vdash R} \quad \frac{\vdash_t Q@(-\infty,0)}{\vdash R}}{\vdash R}$$

A set of equivalencies is used to rewrite formulæ containing operators for which introduction and elimination rules have not been proven in formulæ that contain only operators for which introduction and elimination rules are available:

$$\begin{array}{lcll}
(\neg P) & \equiv & P \Rightarrow \bot & \neg\text{-equiv} \\
(P \Leftrightarrow Q) & \equiv & (P \Rightarrow Q) \wedge (Q \Rightarrow P) & \Rightarrow\text{-equiv} \\
(\exists! x.P(x)) & \equiv & (\exists x.P(x) \wedge (\forall y.P(y) \Rightarrow y = x)) & \exists!\text{-equiv} \\
A {=}\!\rhd\!\rhd B & \equiv & A \Rightarrow (B@[1,1]) & {=}\!\rhd\!\rhd\text{-equiv} \\
A {=}\!\lhd\!\lhd B & \equiv & A \Rightarrow (B@[-1,-1]) & {=}\!\lhd\!\lhd\text{-equiv} \\
A@i,j & \equiv & (A@i) \wedge (A@j) & @,\text{-equiv} \\
A?i,j & \equiv & (A?i) \wedge (A?j) & ?,\text{-equiv} \\
A@i;j & \equiv & (A@i) \vee (A@j) & @;\text{-equiv} \\
A?i;j & \equiv & (A?i) \vee (A?j) & ?;\text{-equiv}
\end{array}$$

Since each TILCO formula is defined with respect to the implicit time, a formula specifies a behavior that holds in different contexts. Hence, this holds even if the interpretation is translated in the temporal domain[1]. Therefore, the following theorem, called the *translation rule*, has been proven:

Theorem 3.1 *If A is a TILCO formula, $\mathcal{I}$ is an interpretation for A and $s,t \in \mathcal{Z}$ then*

$$\mathcal{I}, t \models A \quad \textit{if and only if} \quad \tau(\mathcal{I}, s), t - s \models A.$$

where τ is the temporal translation – see definition 2.1.

Corollary 3.1 *From the previous theorem it follows as a corollary that*

$$\exists \mathcal{I} t.\mathcal{I}, t \models A \quad \textit{if and only if} \quad \forall t.\exists \mathcal{I}.\mathcal{I}, t \models A.$$

As in [46], [44], [47], in TILCO the *generalization rule* cannot be applied to time dependent variables and predicates, thus having a *time generalization* rule (TG). This is due to the implicit model of time. Therefore, a different kind of rule is needed in TILCO to allow generalization over the temporal domain. Thus, as a consequence of TG, the following rule has been proven:

$$\frac{\vdash A}{\vdash A@(-\infty,+\infty)}.$$

[1]This is possible since the specification of temporal constraints is given with respect to events and actions.

This rule states that, if formula A is provable in every time instant, then formula $A@(-\infty, +\infty)$ is true in every time instant. Formula A in the premise of the rule must be provable in *every* time instant, otherwise, from the fact that a formula is true in a given time instant, it could be deduced that the same formula is always true, which is clearly unacceptable. Moreover, it can be easily shown that $A \Rightarrow (A@(-\infty, +\infty))$ is not provable.

3.1 Theorems and Properties

This Section provides a selection of some TILCO's basic properties and demonstrates that each property is a theorem of the TILCO theory. To simplify the demonstrations, application of inference rules has been omitted.

(i) The proofs of the following properties can be easily derived from the definitions of "@" and "?" operators and by considering the logical equivalencies of the predicate calculus:

$$
\begin{array}{lllll}
a) & \vdash \neg(\neg A?i) & \Leftrightarrow & A@i \\
b) & \vdash \neg(\neg A@i) & \Leftrightarrow & A?i \\
c) & \vdash \neg A?i & \Leftrightarrow & \neg(A@i) \\
d) & \vdash (A@i \wedge B@i) & \Leftrightarrow & (A \wedge B)@i \\
e) & \vdash (A@i \vee B@i) & \Rightarrow & (A \vee B)@i \\
f) & \vdash (A?i \vee B?i) & \Leftrightarrow & (A \vee B)?i \\
g) & \vdash (A \wedge B)?i & \Rightarrow & (A?i \wedge B?i)
\end{array}
$$

(ii) Proof of: $\vdash A@i \wedge i \neq \emptyset \Rightarrow A?i$

1.	$\{i \neq \emptyset \wedge A@i\} \vdash i \neq \emptyset \wedge A@i$	Assumption
2.	$\{i \neq \emptyset \wedge A@i\} \vdash A@i$	$\wedge E2$ 1
3.	$\{i \neq \emptyset \wedge A@i\} \vdash_t A@i$	TS 2
4.	$\{i \neq \emptyset \wedge A@i\} \vdash i \neq \emptyset$	$\wedge E1$ 1
5.	$\{i \neq \emptyset \wedge A@i\} \vdash \exists x.x \in i$	4 using various set-theory properties
6.	$\{i \neq \emptyset \wedge A@i\} \vdash x \in i$	EI 5
7.	$\{i \neq \emptyset \wedge A@i\} \vdash_{t+x} A$	$@E$ 3,6
8.	$\{i \neq \emptyset \wedge A@i\} \vdash_t A?i$	$?I$ 6,7
9.	$\vdash_t i \neq \emptyset \wedge A@i \Rightarrow A?i$	DR 8
10.	$\vdash i \neq \emptyset \wedge A@i \Rightarrow A?i$	TG 9

(iii) Proof of: $\vdash A@i,j \Leftrightarrow \neg(\neg A?i;j)$

1.	$\{A@i,j\} \vdash A@i,j$	Assumption
2.	$\{A@i,j\} \vdash A@i \wedge A@j$	@,-equiv. 1
3.	$\{A@i,j\} \vdash \neg(\neg A?i) \wedge A@j$	a) 2
4.	$\{A@i,j\} \vdash \neg(\neg A?i) \wedge \neg(\neg A?j)$	a) 3
5.	$\{A@i,j\} \vdash \neg A?i \vee \neg A?j$	De Morgan 4
6.	$\{A@i,j\} \vdash \neg A?i;j$	?;-equiv. 5
7.	$\vdash A@i,j \Rightarrow \neg A?i;j$	DR 6
8.	$\{\neg A?i;j\} \vdash \neg A?i;j$	Assumption
9.	$\{\neg A?i;j\} \vdash \neg(A?i \vee A?j)$	?;-equiv. 8
10.	$\{\neg A?i;j\} \vdash \neg(A?i) \wedge \neg(A?j)$	De Morgan 9
11.	$\{\neg A?i;j\} \vdash \neg(\neg(\neg A@i)) \wedge \neg(A?j)$	b) 10
12.	$\{\neg A?i;j\} \vdash \neg(\neg(\neg A@i)) \wedge \neg(\neg(\neg A@j))$	b) 11
13.	$\{\neg A?i;j\} \vdash \neg A@i \wedge \neg A@j$	double negation 12
14.	$\{\neg A?i;j\} \vdash \neg A@i,j$	@,-equiv. 13
15.	$\vdash \neg A?i;j \Rightarrow \neg A@i,j$	DR 14
16.	$\vdash (\neg A?i;j \Rightarrow \neg A@i,j) \wedge (A@i,j \Rightarrow \neg A?i;j)$	$\wedge I$ 7,15
17.	$\vdash \neg A?i;j \Leftrightarrow \neg A@i,j$	$\Leftrightarrow$-equiv. 16

The theorem $A?i,j \Leftrightarrow \neg(\neg A@i;j)$ can be proven in a similar way.

(iv) Proof of: $\vdash A@[t_1+t_2, t_1+t_2] \Leftrightarrow A@[t_1,t_1]@[t_2,t_2]$

1.	$\{A@[t_1+t_2,t_1+t_2], t_1 \in [t_1,t_1], t_2 \in [t_2,t_2]\} \vdash A@[t_1+t_2,t_1+t_2]$	assumption
2.	$\{A@[t_1+t_2,t_1+t_2], t_1 \in [t_1,t_1], t_2 \in [t_2,t_2]\} \vdash_t A@[t_1+t_2,t_1+t_2]$	TS 1
3.	$\vdash t_1+t_2 \in [t_1+t_2,t_1+t_2]$	set-theory
4.	$\{A@[t_1+t_2,t_1+t_2], t_1 \in [t_1,t_1], t_2 \in [t_2,t_2]\} \vdash_{t+t_1+t_2} A$	$@E$ 2,3
5.	$\{A@[t_1+t_2,t_1+t_2], t_2 \in [t_2,t_2]\} \vdash_{t+t_2} A@[t_1,t_1]$	$@I$ 4
6.	$\{A@[t_1+t_2,t_1+t_2]\} \vdash_t A@[t_1,t_1]@[t_2,t_2]$	$@I$ 5
7.	$\vdash_t A@[t_1+t_2,t_1+t_2] \Rightarrow A@[t_1,t_1]@[t_2,t_2]$	DR 6
8.	$\{A@[t_1,t_1]@[t_2,t_2], t_1+t_2 \in [t_1+t_2,t_1+t_2]\} \vdash A@[t_1,t_1]@[t_2,t_2]$	assumption
9.	$\{A@[t_1,t_1]@[t_2,t_2], t_1+t_2 \in [t_1+t_2,t_1+t_2]\} \vdash_t A@[t_1,t_1]@[t_2,t_2]$	TS 8
10.	$\vdash t_1 \in [t_1,t_1]$	set-theory
11.	$\vdash t_2 \in [t_2,t_2]$	set-theory
12.	$\{A@[t_1,t_1]@[t_2,t_2], t_1+t_2 \in [t_1+t_2,t_1+t_2]\} \vdash_{t+t_2} A@[t_1,t_1]$	$@E$ 9,11
13.	$\{A@[t_1,t_1]@[t_2,t_2], t_1+t_2 \in [t_1+t_2,t_1+t_2]\} \vdash_{t+t_1+t_2} A$	$@E$ 12,10
14.	$\{A@[t_1,t_1]@[t_2,t_2]\} \vdash_t A@[t_1+t_2,t_1+t_2]$	$@I$ 13
15.	$\vdash_t A@[t_1,t_1]@[t_2,t_2] \Rightarrow A@[t_1+t_2,t_1+t_2]$	DR 14
16.	$\vdash_t \quad (A@[t_1+t_2,t_1+t_2] \Rightarrow A@[t_1,t_1]@[t_2,t_2]) \wedge (A@[t_1,t_1]@[t_2,t_2] \Rightarrow A@[t_1+t_2,t_1+t_2])$	$\wedge I$ 7,15
17.	$\vdash_t A@[t_1,t_1]@[t_2,t_2] \Leftrightarrow A@[t_1+t_2,t_1+t_2]$	$\Leftrightarrow$-equiv. 16
18.	$\vdash A@[t_1,t_1]@[t_2,t_2] \Leftrightarrow A@[t_1+t_2,t_1+t_2]$	TG 17

By using the natural deduction system, many other interesting properties have been derived. For example:

(v) $\{j \subseteq i\} \vdash A@i \Rightarrow A@j$;

(vi) future transitivity: $\vdash j \subseteq i \wedge i = [a, +\infty) \wedge a \geq 0 \Rightarrow (A@i \Rightarrow (A@i)@j)$;

(vii) past transitivity: $\vdash j \subseteq i \wedge i = (-\infty, a] \wedge a \leq 0 \Rightarrow (A@i \Rightarrow (A@i)@j)$;

(viii) linearity: $\vdash ((A@i \Rightarrow B@i)@j) \vee ((B@i \Rightarrow A@i)@j)$.

3.2 Soundness and decidability results

The axiom system proposed for TILCO has been proven to be sound. In fact, to demonstrate its soundness, it is only necessary to prove that each axiom is valid according to the TILCO semantics, and that each deduction rule is sound. In [45] the validity of the axioms and of the soundness of the deductive rules of FOL are demonstrated. Thus, the validity of axioms AX6-AX9 and the soundness of rules $@I$ and $@E$ remain to be proven.

The proof of validity of axioms AX6-AX9 is obtained by using the definitions of operators **until** and **since** given in the TILCO semantics. The demonstration of the soundness of rules $@I$ and $@E$ can be easily constructed by *reductio ad absurdum*.

For example, the proof of soundness of $@E$ is: suppose $@E$ were not sound, then there would be a set of formulæ

$$U = \{A@i, x \in i, A\},$$

such that $A@i$ is valid in t and $x \in i$ is valid, but A is not valid in $x+t$. Since A is not valid in $x+t$, there exists an interpretation $\mathcal{I}$ such that $\mathcal{I}, x+t \not\models A$. Since $A@i$ is valid in t for *any* interpretation, in particular for $\mathcal{I}$, then $\mathcal{I}, t \models A@i$. From TILCO's semantics, this is equivalent to saying that for *any* instant in i (and thus in particular for x) $\mathcal{I}, x+t \models A$ holds, leading to a contradiction. Therefore, the hypothesis that $@E$ was not sound is false, hence $@E$ is proven to be a sound deduction rule.

The same mechanism can be used for demonstrating the soundness of $@I$ rule.

Since TILCO has the expressive power of FOL, validity and satisfiability problems are undecidable in the general case. Nonetheless, the special structure of TILCO formulæ allows the construction of a decision procedure for the validity and satisfiability of a wide set of TILCO formulæ: the only requirement for these formulæ is that non-temporal quantifications must bind only variables whose types have a finite domain.

The demonstration of the validity (satisfiability) of a TILCO formula is equivalent to the validity (satisfiability) of a formula in Presburger arithmetic, which is a decidable problem [48], [49]. The procedure to solve the decision problem for a full Presburger arithmetic has an exponential lower bound, as demonstrated by [50]. One of the most interesting algorithms for solving this problem has been proposed in [48]. Less complex solutions have been proposed for quantifier-free Presburger logic, including uninterpreted predicate and function symbols [51] [49]. In this case, complexity has been reduced to "no worse than exponential". This type of algorithm can be used to manage Skolemited expressions.

The algorithm used for TILCO is only partially based on the above-mentioned approaches and its general complexity is exponential in the worst case. The decision procedure relies on the transformation of TILCO formulæ into FOL formulæ written in prenex conjunctive normal form, after which existential quantifiers are substituted with Skolem constants and functions. For these formulæ the validity and satisfiability problems can be decided by solving a set of parameterized inequalities with a set of constraints for the parameters. In particular, typical TILCO formulæ lead to quite a simple set of arithmetic relations (containing $=$, $\neq$, $<$, $\leq$). The algorithm adopted by our tool is based on the application of a set of heuristics to simplify the theorem through variable elimination and substitution. Heuristics allow formulæ to be rewritten and variables to be eliminated, reducing the theorem to subgoals until few or no subgoals remain to be demonstrated. This process may lead to the direct demonstration of the theorem as well as to a set of linear equalities to be solved by using the classical tactics of Isabelle (see Section 4). The selection of tactics is supported by tacticals or interactively by the user [52], [42] (similar to the approach used in other theorem provers). Therefore, even when the prover does not provide a solution in a reasonable time, the approach aids the user by simplifying and reducing the theorem to smaller goals that can be effectively handled by a human with limited resources.

4 Property Proof and Executability

In order to support the validation of TILCO system specifications, TILCO theory has been formalized in Isabelle [52], [42], which is an automatic theorem proving environment. It allows the definition of new theories and the demonstration of theorems by using either manual or automatic techniques. Isabelle is written in Standard ML [53]; this language is also used for constructing functions and tools for automatic theorem proving.

TILCO theory has been built atop Isabelle/HOL [54], an implementation of Church's High Order Logic [55]. The use of HOL to construct FOL theories has been justified in [56], [57], where this is shown to allow not only the demonstration of theorems *in* the object logic (i.e., TILCO), but also theorems *about* the object logic.

The first steps taken to implement TILCO theory in HOL were to construct a theory defining integer numbers and a theory of intervals. Integer theory has been defined by using equivalence classes over natural numbers as in [58]. This theory provides (i) definitions for all basic arithmetic operations, (ii) a wide set of theorems stating properties of these operations, and (iii) a set of tactics to perform the automatic deduction of theorems in linear arithmetic. Integer theory has been constructed [59] and is presently included among the contributed theories of Isabelle/HOL[2]. Intervals are implemented as connected sets of integers by using the set theory provided by HOL. Interval theory provides constructors for using intervals with the usual mathematical notation with round and square brackets. TILCO theory has been built on the bases of integer and interval theories, and it defines the syntax and semantics of TILCO using Isabelle/HOL as a meta-logic for defining the semantics of its operators. A comprehensive set of theorems regarding TILCO operators has been proven to simplify the construction of theorems either in manual or semi-automatic manner. In particular, the inference rules discussed in the Section 3 have been proven and included in Isabelle's automatic proving tools, thus supporting the automatic proof of medium complexity theorems and assisting the user in demonstrating more complex theorems.

The support offered by TILCO theory in Isabelle/HOL has allowed an absolute degree of confidence in the truth of the theorem proven. This is much safer than using a *pencil and paper* approach, because the use of Isabelle ensures that the demonstrations built are, in fact, correct. In general, with logical approaches, the problem is to demonstrate high-level properties by using only low-level specifications, which usually describe uncorrelated elementary system properties. This can be simplified by using an incremental approach to specification through theorem proving, thus allowing either a top-down or a bottom-up approach:

- Top-Down Approach – A high-level specification is refined into a lower-level specification

[2]Isabelle is located at `http://www.cl.cam.ac.uk/Research/HVG/isabelle.html`.

using theorem proving techniques to validate the refinement, until a detailed specification of the system is obtained;

- Bottom-Up Approach - Theorem proving techniques are used on low-level specifications to prove higher-level properties. This process is repeated until the desired top-level properties are proven.

This approach easily supports the validation of high-level properties, constituting a high-level specification, with respect to the system specification, where intermediate lemmas can also be viewed as intermediate system specifications. This also supports the reuse of specifications of commonly used systems in the specification of a more complex system, by allowing the use of systems that have already been validated by proving their characteristic properties.

Once a TILCO specification of a system is validated against a higher-level TILCO specification by using the TILCO theory in Isabelle/HOL, it can be used as the description of the system itself. A TILCO specification is said to be *causal* if the values of its outputs and auxiliary variables at a given time instant can be determined on the basis of the past history of the system, which includes the past histories of inputs, outputs, and auxiliary variables up to the previous time instant. If a TILCO specification is causal, it can be executed by means of the *TILCO Executor* [43], [60]. If the specification is not causal, then the *TILCO Executor* can be used as a model checker to validate the specification against a complete history of the system, which describes the temporal evolution of inputs, outputs and auxiliary variables in a specific execution.

The classical Tableaux algorithm for FOL [14] has been modified in [61] in order to allow the history checking of specifications written in the logical language TRIO. On the basis of this approach, the execution algorithm underlying the *TILCO Executor* consists of a modified Tableaux algorithm based on three-value logic, instead of the classical approach with binary logic as in [14], [61]. Once the Tableaux for a formula has been built, the tree is iteratively navigated to construct a model for the formula in the current time instant; thus, completing the partial model constituted of the past histories of inputs, outputs and auxiliary variables. If any indeterminacy arises during the execution of a formula, the *TILCO Executor* warns about the problem and also shows the subformulæ that have a undetermined value, but the execution continues as long as the formula can still be evaluated as to its truth. Therefore, the execution of a formula allows the simulation of the system specified by the formula, enforcing an operational approach to the validation of the system specification.

5 Comparison with other Temporal Logics

The availability of a metric for time in a temporal logic is considered one of the most important features for defining quantitative temporal constraints used to specify reactive as well as real-time systems. Among the temporal logics providing a metric for time in its several forms (time intervals, bounded temporal operators, special clock variables, etc.), those based on propositional logic instead of FOL are not expressive enough to describe realistic systems - e.g., [40], EIL [29], RTIL [30], TPTL [15], [22]. In fact, quantification over non-time dependent variables is an abstraction mechanism frequently required for the specification of non-trivial systems.

The most expressive temporal logics providing a metric for time are based on FOL – e.g., RTL [4], MTL [20], TRIO [18], RTTL [16]. A further distinction must be made between temporal logics adopting an implicit or explicit time model. Implicit models produce more concise and readable formulas – e.g., MTL [20], TRIO [18]. Therefore, TRIO and MTL are among the most representative and powerful logics for real-time system specification. Another approach for improving readability may be the graphical representation of temporal specifications; most of the these approaches are based on propositional temporal logic (see GIL [31]).

In this Section we compare TILCO to TRIO [62] and MTL [20] in order to highlight the differences in their conciseness in specifying real-time systems. Many other logics produce specifications structurally similar to TRIO and MTL or have similar operators. However, other logics can be difficult to use in comparison to TRIO, MTL and TILCO, since they are based on elementary operators that lead to the production of overly complex specifications. TRIO and MTL are both based on points and present a sharp distinction between past and future. In contrast, TILCO is an interval temporal logic with a uniform model for time from past to future. These features make TILCO specifications more concise than those in TRIO and MTL, as discussed below.

TILCO provides four elementary temporal operators: **@**, **?**, **since** and **until**. 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). In this paper, basic temporal operators of logics are shown in bold face type. TRIO also provides the quantifiers $\forall$ and $\exists$ on time-dependent variables. In TRIO, on the basis of its operators, several other operators can be defined as parametric predicates. This is also allowed by many temporal logics, including TILCO and MTL. Defining other more specific temporal operators increases the complexity of the logic from a cognitive perspective, since a large number of different functions/operators makes the specification harder to understand. An overabundance of temporal operators does not automatically lead to a greater conciseness and readability. Thus, a comparison of the conciseness of temporal logics must be based on fundamental operators and on their adoption in the context of typical specifications.

Tab.3 shows the most important elementary temporal specifications for TILCO and TRIO; and TRIO-derived temporal operators typically made for using TRIO specifications [62]). Note that specifications in TILCO are more concise than the equivalent TRIO specifications. TILCO is more readable than TRIO even when the new predicates are used as temporal operators in the specification. The verbosity of TRIO is a result of its sharp distinction between past and future, and for its quantifications over time (see the specifications of *Within* and *Always* in

meaning	TILCO	TRIO	TRIO derived
Always Past	$A@(-\infty,0)$	$\forall t(t>0 \rightarrow \mathbf{Past}(A,t))$	$\mathrm{AlwP}(A)$
Always Fut.	$A@(0,\infty)$	$\forall t(t>0 \rightarrow \mathbf{Futr}(A,t))$	$\mathrm{AlwF}(A)$
Always	$A@(-\infty,\infty)$	$\forall t(t>0 \rightarrow \mathbf{Futr}(A,t)) \wedge A \wedge \forall t(t>0 \rightarrow \mathbf{Past}(A,t))$	$\mathrm{Alw}(A)$
Since Weak	$\mathbf{since}(A,B)$	$\forall t''(t''>0 \rightarrow \mathbf{Past}(A,t''))\vee$	
		$\exists t(t>0 \wedge \mathbf{Past}(B,t) \wedge \forall t'(0<t'<t \rightarrow \mathbf{Past}(A,t)))$	$\mathrm{Since}_w(B,A)$
Until Weak	$\mathbf{until}(A,B)$	$\forall t''(t''>0 \rightarrow \mathbf{Futr}(A,t''))\vee$	
		$\exists t(t>0 \wedge \mathbf{Futr}(B,t) \wedge \forall t'(0<t'<t \rightarrow \mathbf{Futr}(A,t)))$	$\mathrm{Until}_w(B,A)$
Lasts	$A@(0,t)$	$\forall t'(0<t'<t \rightarrow \mathbf{Futr}(A,t'))$	$\mathrm{Lasts}(A,t)$
Lasted	$A@(-t,0)$	$\forall t'(0<t'<t \rightarrow \mathbf{Past}(A,t'))$	$\mathrm{Lasted}(A,t)$
Within Past	$A?(-t,0)$	$\exists t'(0<t'<t \wedge \mathbf{Past}(A,t'))$	$\mathrm{WithinP}(A,t)$
Within Fut.	$A?(0,t)$	$\exists t'(0<t'<t \wedge \mathbf{Futr}(A,t'))$	$\mathrm{WithinF}(A,t)$
Within	$A?(-t_1,t_2)$	$\exists t'(0<t'<t_1 \wedge \mathbf{Past}(A,t')) \vee\ A \vee$	
		$\exists t''(0<t''<t_2 \wedge \mathbf{Futr}(A,t''))$	$\mathrm{Within}(A,t_1,t_2)$
Was	$A@(-t_1,-t_2)$	$\mathbf{Past}(\forall t'(0<t'<t_1-t_2 \rightarrow \mathbf{Futr}(A,t')),t_1)$	$\mathbf{Past}(\mathrm{Lasts}(A,t_1-t_2),t_1)$
Will be	$A@(t_1,t_2)$	$\mathbf{Futr}(\forall t'(0<t'<t_2-t_1 \rightarrow \mathbf{Futr}(A,t')),t_1)$	$\mathbf{Futr}(\mathrm{Lasts}(A,t_2-t_1),t_1)$
Could be	$A?(t_1,t_2)$	$\mathbf{Futr}(\neg\forall t'(0<t'<t_2-t_1 \rightarrow \mathbf{Futr}(\neg A,t')),t_1)$	$\mathbf{Futr}(\neg\mathrm{Lasts}(\neg A,t_2-t_1),t_1)$

Table 3: A comparison between TILCO and TRIO on the basis of typical temporal specifications.

Tab.3).

The same behavior can be observed for MTL in Tab.4, where the TRIO examples shown in Tab.3 have been replicated for MTL. In MTL, the elementary set is comprised of four operators: **G** (it is always going to be the case), **F** (at least once in the future), **H** (it has always been the case), and **P** (at least once in the past) [20]. As with TRIO, MTL also defines operators, such as *since* and *until*. As can be seen, MTL is more concise than TRIO, but both make a sharp distinction between past and future, and allow quantification over time-dependent variables; however, a prohibition of this kind of quantification is a necessary condition for the existence of feasible, automated verification mechanisms [41].

Moreover, the adoption of time points instead of intervals also leads to an increase in the complexity of specifications – for instance, specification of *Was*, *Will be* and *Could be* of Tabs.3 and 4 are implemented in TRIO and MTL by using nested operators, while in TILCO only the basic operators have been used. Thus, specifications in MTL are structured much like those in TRIO (in terms of the number of terms and operators). Therefore, TILCO specifications are more readable than those of other logics.

5.1 A more complex example

In this subsection, the comparison is based on the more significant example of a real-time system: an allocator that serves a set of client processes by sharing a resource according to several temporal constraints – [63], [62]. In every time instant and for every process a, the

meaning	TILCO	MTL
Always Past	$A@(-\infty,0)$	$\mathbf{H}A$
Always Future	$A@(0,\infty)$	$\mathbf{G}A$
Always	$A@(-\infty,\infty)$	$\mathbf{H}A \wedge A \wedge \mathbf{G}A$
Since Weak	$\mathbf{since}(A,B)$	$\mathbf{H}A \vee \exists t(t \geqslant 0 \wedge \mathbf{P}_t B \wedge \mathbf{H}_{<t}A)$
Until Weak	$\mathbf{until}(A,B)$	$\mathbf{G}A \vee \exists t(t \geqslant 0 \wedge \mathbf{F}_t B \wedge \mathbf{G}_{<t}A)$
Lasts	$A@(0,t)$	$\mathbf{G}_{<t}A$
Lasted	$A@(-t,0)$	$\mathbf{H}_{<t}A$
Within Past	$A?(-t,0)$	$\mathbf{P}_{<t}A$
Within Future	$A?(0,t)$	$\mathbf{F}_{<t}A$
Within	$A?(-t_1,t_2)$	$\mathbf{P}_{<t_1}A \wedge A \wedge \mathbf{F}_{<t_2}A$
Was	$A@(-t_1,-t_2)$	$\mathbf{P}_{t_1}(\mathbf{H}_{<(t_1-t_2)}A)$
Will be	$A@(t_1,t_2)$	$\mathbf{F}_{t_1}(\mathbf{G}_{<(t_2-t_1)}A)$
Could be	$A?(t_1,t_2)$	$\mathbf{G}_{t_1}(\mathbf{F}_{<(t_2-t_1)}A)$

Table 4: A comparison between TILCO and MTL on the basis of typical temporal specifications.

resource is assigned to process a (gr(a)) if and only if, since the last time the resource was granted (gr(b)) the resource has been released (fr) and

- a requested the resource (rq(a,δ)) and that request has not already expired;
- since the request was issued, the resource has not already been assigned to a;
- there are no a' and δ' such that:
  - $a \neq a'$
  - a' requested the resource (rq(a',δ')) and that request has not already expired;
  - since when the request was issued, the resource has not already been assigned to a';
  - a' requested the resource before a (i.e.: a' did not request the resource after a).

Equation (1) represents the TRIO specification written using only the fundamental operators: **Futr()** and **Past()**, excepted for the presence of the derived operators: the predicate "Alw()". If we express "Alw()" in terms of the basic operators, the specification results at least double in size (in terms of the number of operators) than that of equation (1) (see third line in Tab.3). This type of specification is hard to understand since several quantifications over time are present.

$$
\text{Alw}\left(\forall a.\text{gr}(a) \leftrightarrow \left(\begin{array}{l} \left(\begin{array}{l} \forall t_3(t_3 > 0 \rightarrow \mathbf{Past}(\text{fr}, t_3)) \vee \\ \exists t_1(t_1 > 0 \wedge \mathbf{Past}(\neg\exists b.\text{gr}(b), t_1) \wedge \forall t_2(0 < t_2 < t_1 \rightarrow \mathbf{Past}(\text{fr}, t_2))) \end{array}\right) \wedge \\ \exists t\delta. \left(\begin{array}{ll} \mathbf{Past}(\text{rq}(a,\delta), t) & \wedge \\ t \leq \delta & \wedge \\ t \geq 5 & \wedge \\ \forall t''(0 < t'' < t \rightarrow \mathbf{Past}(\neg gr(a), t)) & \wedge \\ \neg\exists t' a' \delta'. \left(\begin{array}{ll} a' \neq a & \wedge \\ \mathbf{Past}(\text{rq}(a',\delta'), t') & \wedge \\ t' \leq \delta' & \wedge \\ t' \geq 5 & \wedge \\ t' > t & \wedge \\ \forall t'''(0 < t''' < t' \rightarrow \mathbf{Past}(\neg gr(a'), t''')) & \end{array}\right) & \end{array}\right) \end{array}\right)\right) \quad (1)
$$

A more concise version of the same specification can be obtained by defining several specific operators, as shown in equation (2). This requires the reader to learn the semantics of these operators. In [62], a specification for a simpler problem was presented, using an expression a bit more complex than (2), that relied upon quantifications over time variables. For the sake of comparison, equation (2) has been written by the authors to specifically avoid quantifications over time. These are hidden inside the operators (for their definitions see Tab.3). In this case, the specification results more concise. The specification in TRIO can be easily translated into MTL by using Tabs. 3 and 4, yielding an MTL formula with the same structure.

Finally, equation (3) shows the TILCO specification of the same system. It can easily be seen that the TILCO specification is more concise and easier to understand than the others. This is due to: (i) the absence of quantifications over time, (ii) the adoption of a reduced number of temporal operators to express similar concepts, (iii) the lack of a sharp distinction between past and future, and (iv) the adoption of intervals as the elementary time structure instead of the point. Since both TRIO and MTL draw a sharp distinction between past and future the specification contains at least duplicated temporal constraints – two identical sets, one for the past and one for the future. In TILCO, operators **@** and **?** can be used in both past and future, and for writing properties that span a time interval starting in the past and ending in the future (see the specifications for "*within*" in the above tables). Point (iv) leads to the adoption of two nested operators.

TILCO specifications cannot be written in terms of temporal quantifications, since this is not allowed by the language. Therefore, the analyst must write simple and concise specifications.

TRIO version

$$\mathrm{Alw}\left(\forall a.\mathrm{gr}(a) \leftrightarrow \left(\begin{array}{ll} \mathbf{since}_w(\neg\exists b.\mathrm{gr}(b), \mathrm{fr}) & \wedge \\ \exists\delta. \left(\begin{array}{ll} \mathrm{WithinP}(\mathrm{rq}(a,\delta),\delta+1) & \wedge \\ \neg\mathrm{WithinP}(\mathrm{rq}(a,\delta),5) & \wedge \\ \mathbf{since}_w(\neg\mathrm{gr}(a),\mathrm{rq}(a,\delta)) & \wedge \\ \neg\exists a'\delta'. \left(\begin{array}{ll} a' \neq a & \wedge \\ \mathrm{WithinP}(\mathrm{rq}(a',\delta'),\delta'+1) & \wedge \\ \neg\mathrm{WithinP}(\mathrm{rq}(a',\delta'),5) & \wedge \\ \mathbf{since}_w(\neg\mathrm{gr}(a'),\mathrm{rq}(a',\delta')) & \wedge \\ \mathbf{since}_w(\neg\mathrm{rq}(a',\delta'),\mathrm{rq}(a,\delta)) & \end{array}\right) & \end{array}\right) & \end{array}\right)\right) \tag{2}$$

TILCO version

$$\left(\forall a.\mathrm{gr}(a) \leftrightarrow \left(\begin{array}{ll} \mathbf{since}(\neg\exists b.\mathrm{gr}(b), \mathrm{fr}) & \wedge \\ \exists\delta. \left(\begin{array}{ll} \mathrm{rq}(a,\delta)?[-\delta,-5] & \wedge \\ \mathbf{since}(\mathrm{rq}(a,\delta),\neg\mathrm{gr}(a)) & \wedge \\ \neg\exists a'\delta'. \left(\begin{array}{ll} a' \neq a & \wedge \\ \mathrm{rq}(a',\delta')?[-\delta',-5] & \wedge \\ \mathbf{since}(\mathrm{rq}(a',\delta'),\neg\mathrm{gr}(a')) & \wedge \\ \mathbf{since}(\mathrm{rq}(a,\delta),\neg\mathrm{rq}(a',\delta')) & \end{array}\right) & \end{array}\right) & \end{array}\right)\right) @(-\infty,\infty) \tag{3}$$

A formal proof of TILCO's conciseness with respect to that of TRIO and MTL would be difficult, mainly due to the lack of a formal definition of conciseness or readability. Moreover, an examination of the elementary specifications in Tabs. 3 and 4 for TILCO, TRIO and MTL, demonstrates that the typical specifications produced in TILCO use fewer distinct operators than in TRIO and MTL. In addition, some specifications can be written in TRIO and/or MTL only by using nested operators, while simple direct operators are used in TILCO. The TILCO specification is based upon four fundamental operators, while the last TRIO specification includes also derived operators. A greater number of terms are also present in the TRIO/MTL-like formulas in the internal brackets. Thus, complexity increases and conciseness decreases for both TRIO and MTL; and leads to a higher cognitive complexity (or comprehensibility complexity) in programming language (as demonstrated by the validation of several cognitive metrics [64], [65], [66], [67], [68]).

6 Specification Examples

This section provides an example of system specification in order to show TILCO's language capabilities. The system specified is the Alternating Bit Protocol (ABP), which has been proposed in [25], [5], [69] (another very similar protocol has been specified in [20], [19]) and adopted as a classical example for evaluating the expressiveness and conciseness of temporal logics. For the ABP, a high-level specification and an implementation in TILCO are examined, and then the implementation of the ABP is validated against the high-level specification.

6.1 Alternating Bit Protocol

The ABP provides reliable communications over an unreliable communication subsystem. It considers only one message at a time and does not continue to the next message until an acknowledgment of the correct reception of the current message is received. The messages are placed in a packet with a one-bit *sequence number*, thus the name *alternating bit*. For simplicity, the acknowledgment consists of a copy of the packet received. Several packets can be in the communication subsystem simultaneously. Packets can be lost, duplicated or delayed, but not reordered by the communication subsystem. Under these hypotheses, the ABP recovers from every error in the communication subsystem. In the ABP specification below, only a half-duplex protocol (unidirectional communication) is considered as in [25], [5]. The system specification and implementation use the following definitions:

msg is the type of messages that are transmitted by the system;

pkt is the type of packets transmitted over the channel comprised of a message and a Boolean value;

msg_of is a function that extracts the message from a packet;

bool_of is a function that extracts the *sequence number* from a packet;

init is a Boolean constant specifying the initial *sequence number* value;

time_out is an integer constant that specifies the delay between retransmissions of copies of a packet;

fair specifies a fairness relationship between two time-dependent predicates:

$$\mathrm{fair}(A, B) \stackrel{\mathrm{def}}{=} \mathrm{rule}(A\mathbf{?}[0, +\infty)@[0, +\infty) \Rightarrow B\mathbf{?}[0, +\infty)@[0, +\infty))$$

NoReorder is a predicate that is true if the sequences of non-nil values assumed by two expressions are in the same order:

$$\begin{aligned}&\mathrm{NoReorder}(A, B) \stackrel{\mathrm{def}}{=} \forall X.\forall Y.X \neq \mathrm{nil} \wedge Y \neq \mathrm{nil} \Rightarrow\\ &\quad \mathrm{rule}(B = X \wedge B = Y\mathbf{?}(-\infty, 0) \Rightarrow (A = X \wedge A = Y\mathbf{?}(-\infty, 0))\mathbf{?}(-\infty, 0))\end{aligned}$$

NoCreate is a predicate that is true if the occurrence of a non-nil value for an expression has been preceded by the same occurrence for another expression:

$$\mathrm{NoCreate}(A, B) \stackrel{\mathrm{def}}{=} \forall X.X \neq \mathrm{nil} \Rightarrow \mathrm{rule}(B = X \Rightarrow A = X\mathbf{?}(-\infty, 0))$$

TimeOut is a predicate that is true if a predicate has been true for the last δ time instants:

$$\mathrm{TimeOut}(A, \delta) \stackrel{\mathrm{def}}{=} A@(\sim \delta, 0]$$

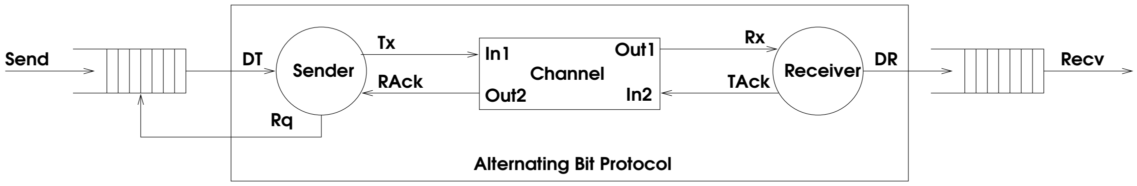

Figure 2: Alternating Bit Protocol schema.

Fig. 2 is a block diagram of the complete system. In the block diagram, the Sender, the Receiver, and the Channel are characterized by the following inputs and outputs:

Rq (output of type **Bool**) is used to request new messages to be transmitted;

DT (input of type **msg**) is used to read a new message to be transmitted to the Receiver;

Tx (output of type **pkt**) is used to send packets over the communication subsystem to the Receiver;

RAck (input of type **pkt**) is used to read acknowledgment packets from the communication subsystem;

DR (output of type **msg**) is used to pass new received messages to environment;

Rx (input of type **pkt**) is used to read packets from the communication subsystem;

TAck (output of type **pkt**) is used to send acknowledgment packets over the communication subsystem to the Sender;

In1(In2) (inputs of type **pkt**) are used to send a packet through the communication subsystem;

Out1(Out2) (outputs of type **pkt**) are used to receive packets from the communication subsystem.

The following assumptions have been made about the environment of the protocol:

- no message transmission is attempted if the ABP does not require a message to be transmitted;
- each message can be uniquely identified;
- each transmission of a message happens in one time instant.

These assumptions are expressed by the formulæ in Tab. 5.

1)	$\text{rule}(\neg\text{Rq} \Rightarrow\!\!\triangleright \text{DT} = \text{nil})$
2)	$\forall m.\text{rule}(\text{DT} = m \Rightarrow \text{DT} \neq m@(-\infty, 0))$
3)	$\text{rule}(\text{DT} \neq \text{nil} \Rightarrow\!\!\triangleright \text{DT} = \text{nil})$

Table 5: Assumptions about the environment of the ABP.

6.1.1 Top-level specification

The TILCO formalization of the ABP requirements, its top-level specification, is given by the following formulæ, which must be satisfied in each time instant:

- every non-nil message passed by the environment to the sender is delivered to the environment by the receiver;
- the ABP cannot create non-nil messages;
- the ABP cannot reorder messages;
- after a message is passed, the ABP eventually requests a new message to be transmitted;
- sometimes the ABP requests a message (i.e., existence of an initial request).

1)	$\text{rule}(\text{DT} = m \wedge m \neq \text{nil} \Rightarrow \text{DR} = m?(0, +\infty))$
2)	$\text{NoCreate}(\text{DT}, \text{DR})$
3)	$\text{NoReorder}(\text{DT}, \text{DR})$
4)	$\text{rule}(\forall m.\text{DT} = m \wedge m \neq \text{nil} \Rightarrow \text{Rq}?(0, +\infty))$
5)	$\text{fact}(\text{Rq})$

Table 6: TILCO top-level specification of the ABP.

This specification is expressed by the formulæ in Tab. 6. It is worth noting that if stronger assumptions can be made regarding the Channel, then formula 4 in Tab. 6 can also be strengthened, as will be shown later in this paper.

In [5] the requirements analysis has been provided only informally by describing the high-level behavior of the system as comprised of the sender, the receiver, and the transmission medium. In our case, the system requirements have also been formalized in TILCO. This methodological approach can be applied by using most of the temporal logics which allow an implicit concept of time, whereas temporal logics with explicit reference to time are too concerned with implementation details to be profitably used — e.g., [4].

6.1.2 Detailed specification

The previous top-level specification has been refined, through a multi-step process, into a detailed specification of the ABP, which constitutes an implementation of the ABP. For each step of the process, the refinement relation between the higher-level and the lower-level specifications has been proven, thus validating the lower-level specification against the higher-level specification. This process ensures that the detailed specification is an implementation of the top-level specification. The implementation constructed contains enough details to be directly executed or operationally validated. In the following discussion, only the final detailed specification is described.

Sender

In the detailed specification of the Sender, the following auxiliary variables are added to the Sender inputs and outputs:

Texp (auxiliary variable of type **Bool**) contains the *sequence number* for the next message;

wait (auxiliary variable of type **pkt**) contains a copy of the packets for which an acknowledgment is waited for.

The detailed specification is defined by the formulæ reported in Tab. 7. Formula 1) imposes the initial conditions of the Sender. The second and the third formulæ show the cases in which the Sender is waiting for a message to be transmitted: 2) if a message is available, the transmission process starts, 3) if no message is available, then the Sender does not change its state. The remaining formulæ specify the Sender's behavior during the message transmission: 4) a correct acknowledgment has been received, then the Sender is ready to accept the next message; 5) the acknowledgment has not been received within the retransmission time, and a new transmission is planned; 6) the acknowledgment has not been received, but the retransmission time has not elapsed yet, so the Sender still waits for the acknowledgment.

1)	$\mathrm{fact}((\mathrm{Tx} = \mathrm{nil} \wedge \mathrm{wait} = \mathrm{nil} \wedge \mathrm{RAck} = \mathrm{nil} \wedge \mathrm{DT} = \mathrm{nil} \wedge \mathrm{Texp} = \mathrm{init} \wedge \mathrm{Rq})@(-\infty, 0])$
2)	$\mathrm{rule}(\mathrm{Rq} \wedge \mathrm{DT} \neq \mathrm{nil} \Rrightarrow \mathrm{Tx} := \mathrm{Pkt}(\mathrm{DT}, \mathrm{Texp}) \wedge \mathrm{wait} := \mathrm{Pkt}(\mathrm{DT}, \mathrm{Texp}) \wedge \mathrm{Texp} := \neg\mathrm{Texp} \wedge \neg Rq)$
3)	$\mathrm{rule}(\mathrm{Rq} \wedge \mathrm{DT} = \mathrm{nil} \Rrightarrow \mathrm{Tx} := \mathrm{nil} \wedge \mathrm{wait} := \mathrm{nil} \wedge \mathrm{Texp} := \mathrm{Texp} \wedge \mathrm{Rq})$
4)	$\mathrm{rule}(\neg\mathrm{Rq} \wedge \mathrm{Texp} \neq \mathrm{bool\_of}(\mathrm{RAck}) \Rrightarrow \mathrm{Tx} := \mathrm{nil} \wedge \mathrm{wait} := \mathrm{nil} \wedge \mathrm{Texp} := \mathrm{Texp} \wedge \mathrm{Rq})$
5)	$\mathrm{rule}(\neg\mathrm{Rq} \wedge \mathrm{Texp} = \mathrm{bool\_of}(\mathrm{RAck}) \wedge \mathrm{TimeOut}(\mathrm{Tx} = \mathrm{nil}, \mathrm{time\_out}) \Rrightarrow \mathrm{Tx} := \mathrm{wait} \wedge \mathrm{wait} := \mathrm{wait} \wedge \mathrm{Texp} := \mathrm{Texp} \wedge \neg\mathrm{Rq})$
6)	$\mathrm{rule}(\neg\mathrm{Rq} \wedge \mathrm{Texp} = \mathrm{bool\_of}(\mathrm{RAck}) \wedge \neg\mathrm{TimeOut}(\mathrm{Tx} = \mathrm{nil}, \mathrm{time\_out}) \Rrightarrow \mathrm{Tx} := \mathrm{nil} \wedge \mathrm{wait} := \mathrm{wait} \wedge \mathrm{Texp} := \mathrm{Texp} \wedge \neg\mathrm{Rq})$

Table 7: TILCO specification of the Sender.

Receiver

In the detailed specification of the Receiver, the following auxiliary variable is added to the Receiver inputs and outputs:

Rexp (auxiliary variable of type **Bool**) which contains the *sequence number* for the next correctly received message.

The detailed specification is defined by the formulæ reported in Tab. 8. In particular, the first formula imposes the initial conditions of the Receiver. The second specifies that every time a packet is received an acknowledgment is sent to the Sender. The third formula represents the case in which a packet with a correct *sequence number* is received; thus, the message is delivered and the *sequence number* updated. The latter specifies that the state of the Receiver does not change if a packet with an incorrect *sequence number* has been received.

1)	$\text{fact}((\text{Rx} = \text{nil} \land \text{TAck} = \text{nil} \land \text{DR} = \text{nil} \land \text{Rexp} = \text{init})@(-\infty, 0])$
2)	$\text{rule}(\text{TAck} := \text{Rx})$
3)	$\text{rule}((\exists m.\text{Rx} = \text{Pkt}(m, \text{Rexp}))\Rightarrow\!\!\triangleright\!\triangleright\text{Rexp} := \neg\text{Rexp} \land \text{DR} := \text{msg\_of}(\text{Rx}))$
4)	$\text{rule}(\neg(\exists m.\text{Rx} = \text{Pkt}(m, \text{Rexp}))\Rightarrow\!\!\triangleright\!\triangleright\text{Rexp} := \text{Rexp} \land \text{DR} := \text{nil})$

Table 8: TILCO specification of the Receiver.

Channel

The communication subsystem has been specified as a couple of identical unidirectional channels. Since only an abstract description of the communication subsystem is known, the Channel specification describes only the high-level properties of the two channels. The detailed specification is defined by the formulæ reported in Tab. 9. The first four formulæ specify that the channels neither create nor reorder messages, respectively. The last two formulæ state the fairness of the two channels, thus allowing delay, duplication, and loss of packets.

1)	$\text{NoCreate}(\text{In1}, \text{Out1})$
2)	$\text{NoCreate}(\text{In2}, \text{Out2})$
3)	$\text{NoReorder}(\text{In1}, \text{Out1})$
4)	$\text{NoReorder}(\text{In2}, \text{Out2})$
5)	$\forall m.\forall v.\text{fair}(\text{In1} = \text{Pkt}(m, v))(\text{Out1} = \text{Pkt}(m, v))$
6)	$\forall m.\forall v.\text{fair}(\text{In2} = \text{Pkt}(m, v))(\text{Out2} = \text{Pkt}(m, v))$

Table 9: TILCO specification of the Channel.

To complete the detailed specification of the ABP, it is only necessary to specify the links between Sender, Receiver and Channel inputs and outputs, by the following formulæ:

- $\text{rule}(\text{Tx} = \text{In1})$;

- rule(Rx = Out1);
- rule(TAck = In2);
- rule(RAck = Out2).

6.1.3 Validation of additional properties

In our detailed specification of the ABP, the fairness property of the channel ensures that, if a packet is sent infinitely often, then it is received infinitely often. This allows us to prove the message delivery, but no time bound can be deduced; consequently, in order to reason about the transmission time, a more detailed version of the channel is needed. The following rules state that if a packet that has already been sent in the past is resent, and if since the previous transmission it has not yet been delivered, then it will be delivered within the next d time units:

$$\text{rule}\left(\begin{array}{l}\forall m.\forall b.\text{In1} = \text{Pkt } m\ b \wedge \text{In1} = \text{Pkt } m\ b\mathbf{?}(-\infty, 0) \wedge \mathbf{since}(\text{In1} = \text{Pkt } m\ b)(\text{Out1} \neq \text{Pkt } m\ b) \Rightarrow \\ \text{Out1} = \text{Pkt } m\ b\mathbf{?}(0, d)\end{array}\right)$$

$$\text{rule}\left(\begin{array}{l}\forall m.\forall b.\text{In2} = \text{Pkt } m\ b \wedge \text{In2} = \text{Pkt } m\ b\mathbf{?}(-\infty, 0) \wedge \mathbf{since}(\text{In2} = \text{Pkt } m\ b)(\text{Out2} \neq \text{Pkt } m\ b) \Rightarrow \\ \text{Out2} = \text{Pkt } m\ b\mathbf{?}(0, d)\end{array}\right)$$

On the basis of this detailed channel specification, the maximum one-way delay of the channel is time_out $+ d$, while the minimum number of packet transmission needed to achieve a one-way delivery is equal to 2. According to this remark, formula 4 in Tab. 6 can be strengthened to:

$$\text{rule}(\forall m.\text{DT} = m \wedge m \neq \text{nil} \Rightarrow \text{Rq}\mathbf{?}(0, k]), \tag{4}$$

where k is a convenient value dependent on the maximum delay and the minimum number of transmissions. In the worst case, in order to receive an acknowledgment, two acknowledgment transmissions are needed which, in turn, require two packet transmissions each. Four packet transmissions require a time duration of $3(\delta+1)$ and the passage of the channel in each direction requires d time units. Two additional time units are used for the beginning and the end of the transmission. Every instance of formula (4) with $k \geq 3\delta + 2d + 5$ can be proven, thus validating the implementation.

In the same way, other high-level properties (both liveness and safeness) have been proven for the system specification.

The complete specification of the above reported example took approximately 25 person-days of work. The work included system analysis, specification of all rules and facts, and the demonstration of all theorems. Once the specification and the demonstrations are given, then Isabelle 94 version 4 is capable of processing the whole specification in about 40 minutes on a SUN Sparc LX workstation with 32Mb of RAM. To give you an idea of the degree of the human interaction required to prove the ABP, the second rule in Tab.6 was validated in about 15 hours (comprising all the proofs of lemmas needed to solve the various subgoals arising in the full demonstration). During the demonstration more than 40 % of the time was spent in solving arithmetical lemmas and theorems to carry out the complete proof. Thus, the formal proof is time consuming but leads to a higher level of assurance than model checking techniques.

The proof time can be greatly reduced by using a more powerful machine. More recently, a new version of Isabelle has been made available. It provides better performance and the automatic tools are more powerful. Thus, the time needed to run the demonstration is reduced. The corresponding new version of TILCO theory is being upgraded.

7 Conclusions

This paper has described the TILCO temporal logic for the specification, validation and verification of real-time systems. It differs from other temporal logics proposed in the literature. TILCO is a first order interval logic that (i) provides a metric for time (thus allowing specification of qualitative and quantitative timing constraints), (ii) presents a linear implicit time model, (iii) adopts a uniform manipulation of intervals from past to future, and (iv) provides decidability for a wide set of formulæ (non-temporal quantifications must bind only variables with types over finite domains). In TILCO no explicit quantification over the temporal domain is allowed. A sound axiomatization has been proposed for TILCO and then used to build a deductive system in natural deduction style. This has been used to prove various TILCO theorems.

Since TILCO is based on FOL and an implicit model of time, it is particularly suitable for requirements analysis and the incremental specification of real-time systems. TILCO supports validation during all phases of the system life-cycle by means of its formalization in the automatic theorem prover Isabelle/HOL. This allows validation for refinement and the proof of general system properties. Moreover, the final operational validation is also supported by the *TILCO Executor*, which allows execution and the model-checking of systems specifications.

Acknowledgments.

The authors would like to thank Prof. G. Bucci for his encouragement and suggestions, L. C. Paulson for his useful suggestions in developing the theory of integers and for providing Isabelle and other related tools, Prof. A. Davis for his valuable suggestions, and Dr. Ing. P. Bellini for his support in verifying the examples in the comparison.

A User-defined data type

In TILCO, three mechanisms for defining new data types are available: ranges over numeric types, type constructors, and datatypes.

Given a numeric type β (i.e., **nat** or **int**), a range type α can be defined as a subtype of type β by using the syntax:

$$\alpha = [x_m, x_M]\beta \quad \text{where } x_m < x_M,$$

which constrains the terms of type α to assume values between x_m and x_M.

Type constructors allow the definition of homogeneous aggregates comprised of entities of simpler types. TILCO defines type constructors for tuple, sets, and lists:

- "$\alpha * \beta$" is the type of the tuple composed of a first element of type α and a second element of type β. A specific tuple is denoted by the expression "(a,b)". More complex tuples can be defined recursively. The operations defined on tuples are the extraction of the first and of the second component of the tuple;

- "α set" is the type of sets comprised of elements of type α. The expression "$\{a, b, c, d\}$" denotes the set composed of elements a, b, c, and d; expression "$\{x.P(x)\}$" denotes the set composed of elements x, such that $P(x)$ holds, where P is a predicate. The usual operations and predicates on sets are defined: $\in$, $\subset$, $\subseteq$, $\cup$, $\cap$, and $\setminus$;

- "α list" is the type of lists comprised of elements of type α. The expression "[ ]" denotes the empty list, "[a,b,c]" denotes the list composed of elements a, b, and c; "$a :: l$" denotes the list constructed by adding the element a at the beginning of list l. The operations defined on lists include: the extraction of the head and tail, the concatenation of two lists, the evaluation of the length of a list, the extraction of the n-th element of a list.

Finally, structured types can be defined by using ML-like datatype declarations:

$$\begin{array}{rcl} \text{datatype TypeVarList Ident} & = & \text{Ident}_1 \text{TypeList}_1 \\ & | & \ldots \\ & | & \text{Ident}_n \text{TypeList}_n \end{array}$$

where TypeVarList is a list of type variables, TypeList_i are lists, possibly empty, of type names comprising previously defined types or type variables in TypeVarList, and Ident_i are distinct identifiers. Recursion inside datatype definitions is allowed through the use of identifiers declared in the previous lines in the datatype definition. Functions and predicates over newly defined datatypes are definable by using pattern-matching definition and primitive recursive functions (if the datatype definition is recursive) by employing Isabelle/HOL facilities for datatype and primitive recursive function definition. Note that if a datatype is used to describe a *message* type variable, an identifier "nil" is usually defined and it is assumed by a variable when no message is available in the evaluation time instant.

Enumerated collections can be defined by using datatype definitions: the datatype is defined by using only identifiers with no TypeList. For example:

$$\text{drink\_type} = \text{coffee}|\text{tea}|\text{milk}.$$

References

[1] G. Bucci, M. Campanai, and P. Nesi, "Tools for Specifying Real-Time Systems," *Journal of Real-Time Systems*, vol. 8, pp. 117–172, March 1995.

[2] A. D. Stoyenko, "The Evolution and State-of-the-Art of Real-Time Languages," *Journal of Systems and Software*, pp. 61–84, April 1992.

[3] A. Pnueli, "The Temporal Logic of Programs," in *18th IEEE FOCS*, 1977.

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

[5] R. L. Schwartz, P. M. Melliar-Smith, and F. H. Vogt, "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), pp. 173–186, Springer Verlag, ACM NewYork, 17-19 Aug. 1983.

[6] R. Gotzhein, "Temporal logic and applications – a tutorial," *Computer Networks and ISDN Systems, North-Holland*, vol. 24, pp. 203–218, 1992.

[7] L. Vila, "A Survey on Temporal Reasoning in Artificial Intelligence," *AI Communications*, vol. 7, pp. 4–28, March 1994.

[8] P. Zave, "An Operational Approach to Requirements Specification for Embedded Systems," *IEEE Transactions on Software Engineering*, vol. 8, pp. 250–269, May 1982.

[9] K. Lano, "Z++, An Object-Oriented Extension to Z," in *Proc. of the 4th Annual Z User Meeting* (J. E. Nicholls, ed.), (Oxford, UK), pp. 151–172, Workshop in Computing, Springer Verlag, 1991.

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

[11] D. Carrington, D. Duke, R. Duke, P. King, G. Rose, and G. Smith, "Object-Z: An Object-Oriented Extension to Z," in *Formal Description Techniques* (S. T. Voung, ed.), Elsevier Science, 1990.

[12] E. H. H. Dürr and J. vanKatwijk, "VDM++: A Formal Specification Language for Object-Oriented Designs," in *Proc. of the International Conference on Technology of Object-Oriented Languages and Systems, TOOLS 7* (G. Heeg, B. Mugnusson, and B. Meyer, eds.), pp. 63–78, Prentice-Hall, 1992.

[13] J. S. Ostroff, "Formal Methods for the Specification and Design of Real-Time Safety Critical Systems," *Journal of Systems and Software*, pp. 33–60, April 1992.

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

[15] R. Alur and T. A. Henzinger, "A really temporal logic," in *30th IEEE FOCS*, 1989.

[16] J. S. Ostroff, *Temporal Logic for Real-Time Systems*. Taunton, Somerset, England: Research Studies Press LTD., Advanced Software Development Series, 1, 1989.

[17] B. C. Moszkowski, *Executing Temporal Logic Programs.* PhD thesis, Cambridge University, 1986.

[18] C. Ghezzi, D. Mandrioli, and A. Morzenti, "TRIO, a logic language for executable specifications of real-time systems," *Journal of Systems and Software*, vol. 12, pp. 107–123, May 1990.

[19] R. Koymans, *Specifying Message Passing and Time-Critical Systems with Temporal Logic.* No. 651, Lecture Notes in Computer Science, Springer-Verlag, 1992.

[20] R. Koymans, "Specifying Real-Time Properties with Metric Temporal Logic," *Real-Time Systems Journal*, vol. 2, pp. 255–299, 1990.

[21] J. F. Allen and G. Ferguson, "Actions and Events in Interval Temporal Logic," tech. rep., University of Rochester Computer Science Department, TR-URCSD 521, Rochester, New York 14627, July 1994.

[22] R. Alur and T. A. Henzinger, "Real Time Logics: complexity and Expressiveness," in *Proc. of 5th Annual IEEE Symposium on Logic in Computer Science, LICS 90*, (Philadelphia, USA), pp. 390–401, IEEE, June 1990.

[23] Z. Manna and A. Pnueli, "Proving Precedence Properties: The Temporal Way," in *Proc. of the 10th Colloquium on Automata Languages and Programming, Lecture Notes in Computer Science LNCS N.154* (J. Diaz, ed.), (Barcelona, Spain), pp. 491–512, Springer Verlag, July 1983.

[24] R. Rosner and A. Pnueli, "A Choppy Logic," in *Proc. of the 1st IEEE Symp. on Logic in Computer Science*, pp. 306–313, IEEE Press, 1986.

[25] R. L. Schwartz and P. M. Melliar-Smith, "From State Machines to Temporal Logic: Specification Methods for Protocol Standards," *IEEE Transactions on Communications*, vol. 30, pp. 2486–2496, Dec. 1982.

[26] J. Halpern, Z. Manna, and B. Moszkowski, "A Hardware Semantics Based on Temporal Intervals," in *Proc. of the 10th Colloquium on Automata Languages and Programming, Lecture Notes in Computer Science LNCS N.154* (J. Diaz, ed.), (Barcellona, Spain), pp. 278–291, Springer Verlag, July 1983.

[27] J. Y. Halpern and Y. Shoham, "A Propositional Modal Logic of Time Intervals," in *Proc. of the 1st IEEE Symp. on Logic in Computer Science*, pp. 274–292, IEEE Press, 1986.

[28] P. Ladkin, "Models of Axioms for Time Intervals," in *Proc. of the 6th National Conference on Artificial Intelligence, AAAI'87*, (USA), pp. 234–239, 1987.

[29] P. M. MelliarSmith, "Extending Interval Logic to Real Time Systems," in *Proc. of Temporal Logic Specification United Kingdom, (B. Banieqbal, H. Barringer, A. Pnueli, eds)*, pp. 224–242, Springer Verlag, Lecture Notes in Computer Sciences, LNCS 398, April 1987.

[30] R. R. Razouk and M. M. Gorlick, "A Real-Time Interval Logic for Reasoning About Executions of Real-Time Programs," in *Proc. of the ACM/SIGSOFT'89, Tav.3*, pp. 10–19, ACM Press, Dec. 1989.

[31] L. K. Dillon, G. Kutty, L. E. Moser, P. M. Melliar-Smith, and Y. S. Ramakrishna, "A Graphical Interval Logic for Specifying Concurrent Systems," *ACM Transactions on Software Engineering and Methodology*, vol. 3, pp. 131–165, April 1994.

[32] J. F. Allen, "Maintaining Knowledge about Temporal Intervals," *Communications of the ACM*, vol. 26, pp. 832–843, Nov. 1983.

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

[34] C. Stirling, "Comparing Linear and Branching Time Temporal Logics," in *Proc. of the International Conference on Temporal Logic in Specification* (B. Banieqbal, H. Barringer, and P. Pnueli, eds.), (Altrincham, UK), pp. 1–20, Springer Verlag, LNCS n.398, 8-10 April 1987.

[35] M. Fisher and R. Owens, "An Introduction to Executable Modal and Temporal Logics," in *Executable Modal and Temporal Logics: Proceedings of the IJCAI '93 Workshop, Chamberry, France, August 1993* (M. Fisher and R. Owens, eds.), pp. 1–20, Lecture Notes in Artificial Intelligence, Springer Verlag LNCS 897, 1995.

[36] M. Felder and A. Morzenti, "Validating Real-Time Systems by History-Checking TRIO Specifications," in *Proc. of 14th International Conference on Software Engineering*, (Melbourne, Australia), pp. 199–211, IEEE press, ACM, 11-15 May 1992.

[37] J. S. Ostroff and W. Wonham, "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, Dec. 1987.

[38] G. Bucci, M. Campanai, P. Nesi, and M. Traversi, "An Object-Oriented Dual Language for Specifying Reactive Systems," in *Proc. of IEEE International Conference on Requirements Engineering, ICRE'94*, (Colorado Spring, Colorado, USA), 18-22 April 1994.

[39] G. Bucci, M. Campanai, P. Nesi, and M. Traversi, "An Object-Oriented CASE Tool for Reactive System Specification," in *Proc. of 6th International Conference on Software Engineering and Its Applications (sponsored by: EC2, CXP, CIGREF, and SEE)*, (Le CNIT, Paris la Defense, France), 15-19 Nov. 1993.

[40] T. A. Henzinger, Z. Manna, and A. Pnueli, "Temporal Proof Methodologies for Real-Time Systems," in *Proc. of the 18th ACM Symposium on Principles of Programming Languages*, pp. 353–366, ACM, 1991.

[41] R. Alur and T. A. Henzinger, "Real-Time Logics: Complexity and Expressiveness," tech. rep., Dept. of Comp. Science and Medicine STAN-CS-90-1307, Stanford University, Stanford, California, USA, March 1990.

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

[43] R. Mattolini, *TILCO: a Temporal Logic for the Specification of Real-Time Systems (TILCO: una Logica Temporale per la Specifica di Sistemi di Tempo Reale).* PhD thesis, Dipartimento di Sistemi e Informatica, University of Florence, Italy, 1996.

[44] M. Felder, D. Mandrioli, and A. Morzenti, "Proving Properties of Real-Time Systems Through Logical Specifications and Petri Net Models," tech. rep., Politecnico di Milano, Dipartimento di Elettronica e Informazione, 91-072, Piazza Leonardo da Vinci 32, Milano, Italy, 1991.

[45] R. E. Davis, *Truth, Deduction, and Computation: Logic and Semantics for Computer Science.* New York: Computer Science Press, 1989.

[46] J. Thistle and W. M. Wonham, "Control Problems in a Temporal Logic Framework," *International Journal of Control*, vol. 44, no. 4, 1986.

[47] H. B. Henderton, *A Mathematical Introduction to Logic.* New York: Academic Press, 1972.

[48] D. C. Cooper, "Theorem Proving in Arithmetic without Multiplication," *Machine Intelligence, American Elsevier*, vol. 7, pp. 91–99, 1972.

[49] W. W. Bledsoe, "A New Method for Proving Certain Presburger Formulas," in *4th Int'l Joint Conf. on Artificial Intelligence*, pp. 15–21, September 1975.

[50] M. J. Fischer and M. O. Rabin, "Super-exponential Complexity of Presburger Arithmentic," in *Complexity of Computation* (R. M. Karp, ed.), (Providence), pp. 27–31, R.I., Mathematical Society, 1974.

[51] R. E. Shostak, "A Practical Decision Procedure for Arithmetic with Function Symbols," *Journal of the ACM*, vol. 26, pp. 351–360, Apr. 1979.

[52] L. C. Paulson, "Isabelle: The Next 700 Theorem Provers," in *Logic and Computer Science* (P. Ochifreddi, ed.), pp. 361–386, Academic Press, 1990.

[53] L. C. Paulson, *ML for the Working Programmer.* Cambridge University Press, 1991.

[54] L. C. Paulson, "Isabelle's Object-Logics," Tech. Rep. 286, Computer Laboratory, University of Cambridge U.K., 1994.

[55] A. Church, "A formulation of the simple theory of types," *JOURNAL of Symbolic Logic*, no. 5, pp. 56–68, 1940.

[56] J. P. Bowen and M. J. C. Gordon, "Z and HOL,," in *Z User Workshop, Cambridge 1994* (J. P. Bowen and J. A. Hall, eds.), pp. 141–167, Workshops in Computing Series, Springer-Verlag, 1994.

[57] J. P. Bowen and M. J. C. Gordon, "A Shallow Embedding of Z in HOL," *Information and Software Technology*, vol. 37, pp. 269–276, May/June 1995.

[58] J. Harrison, "Constructing the real numbers in HOL," Tech. Rep. HOL, University of Cambridge Computer Laboratory New Museums Site, Pembroke Street, Cambridge, CB2 3QG, England, 1993.

[59] R. Mattolini and P. Nesi, "Formalizing the Integers in Isabelle/HOL," Tech. Rep. DSI-RT 28/94, Dipartimento di Sistemi e Informatica, Università degli Studi di Firenze, 1994.

[60] A. Giotti, R. Mattolini, and P. Nesi, "Executing TILCO Specification with Tableaux," Tech. Rep. DSI-RT 31/95, Dipartimento di Sistemi e Informatica, Università degli Studi di Firenze, 1995.

[61] D. Mandrioli, S. Morasca, and A. Morzenti, "Functional test case generation for real-time systems," tech. rep., Politecnico di Milano, Dipartimento di Elettronica e Informazione, 91-072, Piazza Leonardo da Vinci 32, Milano, Italy, 1992.

[62] M. Felder and A. Morzenti, "Validating Real-Time Systems by Hystory-Checking TRIO Specifications," *ACM Transactions on Software Engineering and Methodology*, vol. 3, pp. 308–339, Oct. 1994.

[63] H. Barringer, M. Fisher, D. Gabbay, G. Gough, and R. Owens, "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), Springer Verlag, LNCS n.430, June 1989.

[64] R. A. McCauley and W. R. Edwards, "Analysis and Measurement Techniques for Logic-Based Languages," in *Proc. of the 2nd Euromicro Conference on Software Maintenance and Reengineering*, (Florence, Italy), IEEE Press, 8-11 March 1998.

[65] S. N. Cant, D. R. Jeffery, and B. Henderson-Sellers, "A Conceptual Model of Cognitive Complexity of Elements of the Programming Process," tech. rep., University of New South Wales, Information Technology Research Centre, n.57, New South Wales 2033, Australia, Oct. 1991.

[66] H. Zuse, *Software Complexity: measures and methods.* Berlin, New-York: Walter de Cruyter, 1991.

[67] S. D. Conte, H. E. Dunsmore, and V. Y. Shen, *Software Engineering Metrics and Models.* Benjaming/Cumminggs, Publ. Co., 1986.

[68] P. Nesi and M. Campanai, "Metric Framework for Object-Oriented Real-Time Systems Specification Languages," *The Journal of Systems and Software*, vol. 34, pp. 43–65, 1996.

[69] I. Suzuki, "Formal Analysis of the Alternating Bit Protocol by Temporal Petri Nets," *IEEE Transactions on Software Engineering*, vol. 16, pp. 1273–1281, Nov. 1990.

Contents

Authors' Biographies

Paolo Nesi was born in Florence, Italy, in 1959. He received his DrEng degree in electronic engineering from the University of Florence, Italy. He received the Ph.D. degree from the University of Padoa, Italy, in 1992. In 1991, he was a visitor at the IBM Almaden Research Center, CA, USA. Since 1992, he is with the Dipartimento di Sistemi e Informatica, where he is Associate Professor for the University of Florence, Italy. He is active on several research topics: formal methods, object-oriented technology, real-time systems, system assessment, physical models, parallel architectures. He has been program chair or co-chair of some international conferences. He is a member of program committee of several international Conferences, among them: IEEE ICECCS, IEEE METRICS, EUROMICRO CSMR, REF. He has been Guest Editor of special issues of international journals and an editorial board member of the Journal of Real-Time Imaging, Academic Press; and of a book series of CRC. He is the author of more than 100 technical papers and 4 books. Nesi has been the project responsible and coordinator of several ESPRIT projects and holds the scientific responsibility at the CESVIT (High-Tech Agency for technology transfer) for object-oriented technologies and HPCN. He is a member of IEEE, AIIA, and TABOO.

Riccardo Mattolini was born in Florence, in 1966. He received his DrEng degree in Electronic Engineering and the Ph.D. from the University of Florence, Italy. He has been a visitor of University of Colorado, in Colorado Springs, USA, in 1995. His research interests include formal languages and software engineering tools for real-time systems specification, theorem proving applied to software verification, operational languages for distributed and real-time systems. Mattolini is presently a technical consultant in Hewlett-Packard Italy. He is a member of the IEEE.