A finite-domain semantics for testing temporal logic specifications

Alberto Coen-Porisini, Matteo Pradella, and Pierluigi San Pietro

Dipartimento di Elettronica ed Informazione, Politecnico di Milano
P.za Leonardo da Vinci, 32, 20133 Milano, Italia
{coen, pradella, sanpietr}@elet.polimi.it

Abstract. A method to achieve executability of temporal logic specifications is restricting the interpretation of formulas over finite domains. This method was proven to be successful in designing testing tools used in industrial applications. However, the extension of the results of these tools to infinite, or just larger, domains, requires an adequate definition of a "finite-domain semantics". Here we show the need for correcting previous semantics proposals, especially in the case of specifications of real-time systems which require the use of bounded temporal operators. We define a new semantics for the TRIO language, easily extendible to other linear metric temporal logic languages and show its adequateness on various examples.

1 Introduction

The use of formal executable specifications has many advantages: by executing formal specifications it is possible to observe the behavior of the specified system and check whether they capture the intended functional requirements. This kind of analysis, called *specification testing*, increases the confidence in the correctness of the specification in much the same way as testing a program may increase the confidence on its reliability, assessing the adequacy of the requirements before a costly development takes place. Moreover, execution may allow the generation of test data, that can be used for functional testing, that is for checking the correctness of the implementation against the specification [8,9].

A specification formalism very suitable to real-time systems is Linear Temporal Logic with a metric of time [5]. Such a formalism allows one to express complex temporal conditions and properties in a precise, quantitative way, while its denotational style allows one to abstract from implementation details until the beginning of the development phase. However, specifications written using a first-order temporal logic with metric are in general not decidable.

Various proposals have appeared in the literature to define less powerful and less expressive, but decidable, temporal logics [1]. As a consequence many properties cannot be expressed in such languages. A different approach to overcome the undecidability problem consists in using finite domains to interpret (hence, test) first-order specifications [7]. The result is a decidable language as expressive as a first-order logic, although as powerful as a propositional logic.

Executability is achieved by defining a model-theoretic semantics (i.e., an interpretation schema) that, for any formula, builds its possible *models* (i.e., assignments of values to variables and predicates such that the formula evaluates to true), and by exploiting the idea of *finite approximation of infinite domains*. Original interpretation domains, which are usually infinite, are replaced by finite approximations thereof. For instance, the set of integers is replaced by the range 0..100.

In this way, every decision problem becomes decidable even though there is no *a priori* guarantee that the results obtained on the finite domain coincide with the theoretical results that would be obtained on the infinite domain. In practice, however, we may often rely on this type of prototyping, especially if the domains are large enough to contain all the "relevant facts" about the system under analysis, on the basis of the following considerations:

- Non-terminating reactive systems often have periodical behaviors. Thus it usually suffices to analyse them for a time period of, for instance, twice their periodicity.
- Using some common sense and experience one can tell whether all "relevant facts" about a system, whose dynamic behavior is in the order of magnitude of seconds, have been generated after having tested it for several hours.
- One may try several executions with different time domains of increasing cardinality. If the results do not change, one can infer that they will not change for larger domains.

Clearly, this approach is based on the assumption that it is possible to significantly evaluate formulas on a finite time domain, while a specifier usually makes the natural assumption of an *infinite* time domain. Hence, the finiteness is only a "trick" to enable specification execution and therefore it is crucial to provide a finite-domain semantics such that the results obtainable on finite histories may be easily extended to infinite behaviors of the system.

Execution algorithms based on a finite time semantics have been successfully designed for the TRIO language [4] and have been tested in the context of several industrial applications [2].

Various proposals of finite-domain semantics have appeared in the literature, for both TRIO and other temporal logic languages. In [5], a conventional $false$ (or $true$) value is given to every formula (or part of a formula) whose evaluation time does not belong to the time domain. Very early it was recognised that this resulted in a very counterintuitive semantics. In [3] the language has two temporal distance operators: a strong operator $\triangle$ and a weak one $\triangledown$. $\triangle tF$ is true iff there exists a time instant whose distance from the current one equals t, in which F is true , while $\triangledown tF$ is true iff there exists a time instant whose distance from the current one equals t, in which F is true, or if there is no such t. Using this approach the specification must be written taking into account the finite domains from the beginning. Moreover, the use of two different distance operators has proved to be confusing for most users.

The best proposal up to now is the model-parametric semantics (MPS) presented in [7]. MPS correctly interpretates many cases of practical interests that

are not dealt with adequately by the other proposals. However, in this paper we show that MPS causes formulas with bounded temporal operators, which impose upper or lower bounds on the occurrence of events, to become counter-intuitive and has also some other minor problems. These problems in the interpretation of formulas over finite domains limit the validity and the use of the tools for executing specifications and hence may seriously hamper the validation phase. This is especially true for real-time systems since most of them require explicit time bounds, that is their specification must use bounded temporal operators. It is then of the outmost importance to define the semantics of such operators in the most general and intuitive way.

This paper presents a new semantics for the finiteness problem by modifying MPS. In particular, it provides a different semantics for the bounded temporal operators, along with some minor changes to the original definition in order to deal correctly with all temporal operators. Algorithms have been derived and already implemented from our definitions, and various experiments have been performed, showing the validity of our approach. It must be noticed that although our work has been carried out for the TRIO language, it can be easily extended to other temporal logic languages.

2 An Overview of TRIO

TRIO is a first order temporal logics that supports a linear notion of time: the *Time Domain* T is a numeric set equipped with a total order relation and the usual arithmetic relations and operators. The time domain represents the set of instants where a TRIO formula may be evaluated. Another special domain is the *distance domain* ΔT, a numeric domain composed of the distances between instants of the time domain.

TRIO formulas are constructed in the classical inductive way, starting from terms and atomic formulas. Besides the usual propositional operators and the quantifiers, one may compose TRIO formulas by using a single basic modal operator, called $Dist$, that relates the *current time*, which is left implicit in the formula, to another time instant: the formula $Dist(F, t)$, where F is a formula and t a term indicating a time distance, states that F holds at a time instant at t time units from the current instant.

For the sake of convenience, TRIO items (variables, predicates, and functions) are divided into time-independent (TI) ones, i.e., those whose value does not change during system evolution (e.g., the altitude of a reservoir) and time-dependent (TD) ones, i.e., those whose value may change during system evolution (e.g., the water level inside a reservoir).

TRIO formulas are evaluated on a *history*, that is a sequence of events. A model or behavior of a formula F is a history h such that F evaluates to *true* on h.

Several *derived temporal operators* can be defined from the basic $Dist$ operator through propositional composition and first order quantification on variables representing a time distance. A sample list of such operators is given below,

along with their definition and a short explanation. The variable d is over the distance domain ΔT.

- $Lasts(F,t) \stackrel{\triangle}{=} \forall d(0 < d < t \rightarrow Dist(F,d))$, i.e. F holds over a period of length t in the future.
- $Lasted(F,t) \stackrel{\triangle}{=} \forall d(0 < d < t \rightarrow Dist(F,-d))$, i.e. F held over a period of length t in the past.
- $Alw(F) \stackrel{\triangle}{=} \forall d Dist(F,d)$, i.e. F always holds.
- $SomF(F) \stackrel{\triangle}{=} \exists d(d > 0 \wedge Dist(F,d))$, i.e. sometimes in the future F will hold.
- $WithinF(F,t) \stackrel{\triangle}{=} \neg Lasts(\neg F,t)$, i.e. F will occur within t time units.
- $Until(F1,F2) \stackrel{\triangle}{=} \exists d(d > 0 \wedge Lasts(F1,d) \wedge Dist(F2,d))$, i.e. $F1$ will last until $F2$ occurs.
- $Becomes(F) \stackrel{\triangle}{=} F \wedge Dist(\neg F,-1)$, i.e. F holds at the current instant but did not hold in the previous instant.

The traditional operators of linear temporal logics can be easily obtained as TRIO derived operators. For instance, $SomF$ corresponds to the *Eventually* operator of temporal logic. Moreover, it can be easily shown that the operators of several versions of temporal logic (e.g., interval logic) can be defined as TRIO derived operators. This argues in favour of TRIO's generality since many different logic formalisms can be described as particular cases of TRIO.

In what follows we provide two simple examples of a TRIO specification. Even though they are very simple they will be used in Section 3 to highlight the problems related to the current definition of TRIO's semantics.

Example: A transmission line Consider a simple transmission line, that receives messages at one end and delivers them at the opposite end with a fixed delay (e.g., 5"). The arrival of a message is represented by the time dependent predicate *in*, while its delivery is represented by predicate *out*. The following formula expresses that every received message is delivered after exactly 5 seconds from its arrival.

- (TL) $Alw(in \leftrightarrow Dist(out,5))$.

The use of the equivalence operator '$\leftrightarrow$' ensures that no received message gets lost and no spurious message (i.e., an output without an input) is emitted. Figures 1 and 2 show examples of histories on which formula (TL) is verified.

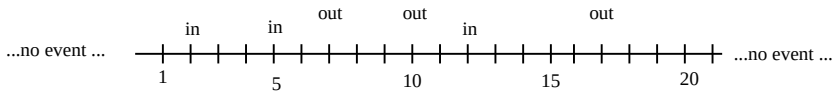

Fig. 1. A history for the transmission line example, representing a finite behavior

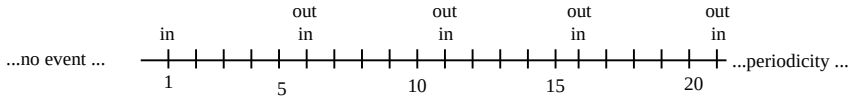

Fig. 2. A periodic (infinite) behavior for the transmission line example, where an in occurs forever exactly every 5 time instants, starting from instant 1

Example: A lamp with a timer Let us consider a lamp with a timer and a switch. When the lamp is off, pushing the button of the switch turns the lamp on. When the lamp is on, pushing the button turns the lamp off; moreover if the button is not pushed the lamp is turned off anyway by a timer after 4 time units. In the specification, the state of the lamp is modelled by the time dependent predicate *on*, which is true iff the lamp is on; the event of pushing the button is modelled by the time dependent predicate *push*. Finally, another time dependent predicate *timeout*, models the timer of the lamp. The specification of this system is the formula $Alw(A1 \wedge A2 \wedge A3)$, where $A1$, $A2$ and $A3$ are the following formulas:

- (A1) $timeout \leftrightarrow Lasted(on, 5)$.
- (A2) $Becomes(on) \leftrightarrow push \wedge Dist(\neg on, -1)$.
- (A3) $Becomes(\neg on) \leftrightarrow (push \wedge Dist(on, -1) \vee timeout)$.

(A1) defines the timeout: the light has been on during the last 4 time units[1].

(A2) states that the lamp becomes on iff the button is pushed and the light was off.

(A3) states that the lamp becomes off iff either there is a timeout or the button is pushed while the light was on.

Let us consider the behavior shown in Figure 3 (where every predicate is false unless explicitly indicated): the button is pushed a first time at instant 3, and thus the light is on from instants 3 to 6, since the *timeout* occurs at instant 7. The light stays off until the next *push* (instant 10), remains on from 10 to 12, when another *push* occurs (instant 13), which finally turns the light off.

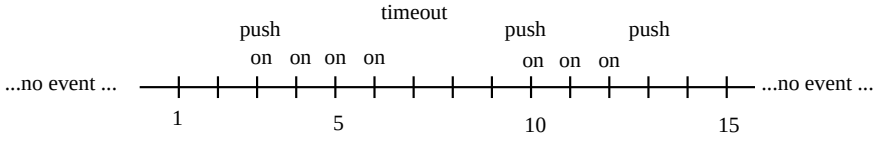

Fig. 3. A history for the timed lamp example

[1] The definition of *Lasted* does not consider the current time instant, hence $Lasted(on, 5)$ requires *on* to be true in the previous four time instants.

2.1 The TRIO environment

The practical usefulness of a specification language depends not only on its inherent qualities (rigor, ease and naturalness of use, generality, etc.), but also on the availability of tools that favour its use. Thus, we are presently developing a complete environment of TRIO-based tools, that includes:

- An interactive, graphical editor for TRIO. The editor supports the incremental writing of TRIO-based specifications and histories.
- A TRIO interpreter which supports specification prototyping and verification of their properties such as consistency. Different operating modes allow the designer to manage a trade-off between generality and complexity.
- A test case generator, which allows the (semi)automatic i.e., user-driven generation of functional test cases associated with a given TRIO specification against which the designer can check the implementation. The core of the tool is the interpretation mechanism, tailored to the specialised use of test cases production, which generate models compatible with the given specifications.

The interpreter and the test case generator are based on the definition of a finite-domain semantics for TRIO.

3 TRIO's formal semantics: problems and solutions

Let us consider formula (TL) of example 1 (the transmission line), and the history depicted in Figure 1 restricted to the instants 1..20. This is certainly an acceptable behavior of the system. Notice that developing a finite-domain semantics adequate to this example is fairly easy; for instance, by providing a conventional evaluation to false for everything lying outside the time domain.

However, consider the history of Figure 2, again restricted to the instants 1..20, which is reported in Figure 4.

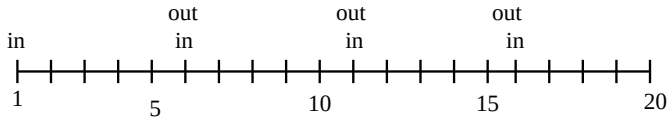

Fig. 4. The finite restriction of history of Figure 2 to the domain 1..20

In this case, *in* is true at instant 16 and there is no corresponding *out*, since it would occur at instant 21, which is outside the time domain. However, also this (finite) behavior should be considered acceptable because the *in* at instant 16 is a border event: it could be followed by an *out* at instant 21, that is there exists at least one infinite history containing all the events of Figure 4 which satisfies (TL). Obviously, there are also infinite histories that include the history

of Figure 4 which do not satisfy (TL) (e.g. a history in which there is no *out* at instant 21).

Therefore, a finite-domain semantics should consider the history of Figure 4 as a model of (TL). The use of conventional truth values, however, does not work since it would conventionally assume that at instant 21 *out* is false (assuming a conventional true value outside the temporal domain would even worsen the situation). Enlarging or restricting the time domain does not solve this problem since: if the instant 21 is included in the time domain then also the instant 26 should be included and so on.

In order to enable the execution of TRIO specifications a model parametric semantics MPS was defined. The MPS may refer to any finite or infinite time domain T, the finite case being considered an approximation of the infinite one. If the finite time domain is large enough to include all relevant events, the corresponding finite history is included in an infinite behavior of the system.

As a consequence the size of the time domain is quite important: if it is too small some relevant event may not be included, and thus the finite restriction of an infinite history may not be meaningful. For instance, a specification such as $Som(in)$ is verified whenever there exists at least one occurrence of in. Every finite history in which no in occurs can hardly be considered to verify $Som(in)$; however, any such history can be extended to include an occurrence of in, and hence it is a sub-history of a behavior of the system.

The basic idea of MPS consists in not evaluating a formula in those instants in which the truth of the formula depends on what may or may not occur outside the time domain. For instance, the meaning of the Alw operator becomes: $Alw(A)$ is true iff A holds in every instant in which A may be evaluated. $Som(A)$ is true iff there exists an instant in which A can be evaluated and holds. If A cannot be evaluated in any instant, then $Alw(A)$ and $Som(A)$ are considered meaningless.

Hence, according to MPS, formula (TL), evaluated on the history of Figure 4 becomes true, since the subformula $in \leftrightarrow Dist(out, 5)$ is true where it can be evaluated, that is on the range 1..15. The truth value of the formula is not checked in 16, since $Dist(out, 5)$ cannot be evaluated.

To better understand MPS and its problems in what follows we summarize its formal definition given in [7].

3.1 MPS Formal Definition

For the sake of simplicity we consider only formulas where all variables are of the type distance domain (ΔT), which in MPS is interpreted as the interval $-|T|+1..|T|-1$, and there are no time dependent functions or constants.

A quantifier $\forall x$ in a formula of type $\forall x A$ is restricted to those values $a \in \Delta T$ such that A_x^a (the formula obtained from A by replacing every occurrence of x with the value a) can be evaluated without referencing time instants outside the time domain. This can be obtained by defining a function $Eval$ that associates every formula with the subset of T on which it can be evaluated. The definition of $Eval$ is:

1. $Eval(P) = T$, for an atomic formula P.
2. $Eval(\neg A) = Eval(A)$.
3. $Eval(A \wedge B) = Eval(A) \cap Eval(B)$.
4. $Eval(Dist(A, t)) = \{i \in T | i + t \in Eval(A)\}$.
5. $Eval(\forall x A) = \bigcup_{a \in \Delta T} Eval(A_x^a)$.

A formula A is said to be not evaluable iff $Eval(A) = \emptyset$, that is it cannot be evaluated in any instant. In this case, the formula A is considered meaningless.

Notice that the evaluation of formulas following MPS differs from traditional evaluation only when quantifiers are involved. In fact, if a formula such as $\forall x A$ can be evaluated, (i.e., $Eval(\forall x A) \neq \emptyset$), then its truth value in an instant i is true if A_x^a is true in i for every $a \in \Delta T$ such that i belongs to $Eval(A_x^a)$, it is false otherwise.

3.2 Problems of MPS

While we believe that the general idea behind the MPS is very appealing, its definition is not completely satisfactory. Its main problems are discussed in what follows and concern the characterization of the distance domain ΔT, the treatment of the bounded operators and the semantics of propositional operators.

The distance domain ΔT Let us consider the timed lamp example and the restriction of the history depicted in Figure 3 to the time domain 1..15, as shown in Figure 5.

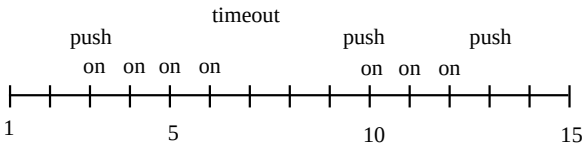

Fig. 5. A restriction of history 3 to 1..15

This behavior is intuitively correct and obviously includes every relevant event. However, according to MPS the subformula (A1), $timeout \leftrightarrow Lasted(on, 5)$, is false at instant 1. In fact, the definition of $Lasted(on, 5)$, is $\forall d(0 < d < 5 \rightarrow Dist(on, -d))$, where d is a variable in the distance domain ΔT = -14..14. Thus, at instant 1 d can assume any value in the range -14..0[2], and the condition $0 < d < 5$ is false for every d. As a consequence, $Lasted(on, 5)$ is true in 1, while $timeout$ is not.

The problem does not disappear by extending the time domain: there is always a left border where (A1) can be false. Other similar counterexamples can be built for other TRIO operators, such as $Since$ and $Until$, and are very puzzling for most users of the tools based on MPS.

[2] The values greater than 0 are ruled off since $Dist(on, -d)$ cannot be evaluated

The problem arises from the use of non positive values in the distance domain $\Delta T = -|T| + 1..|T| - 1$, that may create undesired border effects. It can be solved by defining ΔT as $1..|T| - 1$. In this case, the subformula $\forall d(0 < d < 5 \rightarrow Dist(on, -d))$ becomes not evaluable at instant 1 and therefore instant 1 is ignored when evaluating (A1).

Bounded operators Let us further restrict the time domain of the previous example to the range 4..15 (Figure 6).

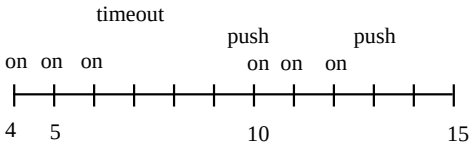

Fig. 6. A behavior for the timed lamp, where (A1), (A2) and (A3) are false

The history describes what can still be considered an intuitively acceptable behavior for the specification (A1): there is a border effect in the instants from 4 to 7, where the lamp has been on for less than 4 instants. Therefore we cannot know whether timeout should actually hold at instant 7. However, we expect the axiom (A1) to hold, because the left border 4..7 should not be considered for evaluation; at most, we could accept that the specification is not evaluable, thus signalling that we should have chosen a larger domain including at least instant 3. Unfortunately, also in this case according to MPS formula (A1) is false at instant 6 (and also at instant 5), and thus the specification $Alw(A1 \wedge A2 \wedge A3)$ does not hold. In fact, looking at formula $\forall d(0 < d < 5 \rightarrow Dist(on, -d))$, i.e. $Lasted(on, 5)$, we notice that at instant 6, d can have at least values 1 and 2 (both if $\Delta T = -|T| + 1..|T| - 1$ or $\Delta T = 1..|T| - 1$). Hence, $Lasted(on, 5)$ holds at instant 6, because $Dist(on, -1)$ and $Dist(on, -2)$ hold. But *timeout* is false at 6.

The problem is that MPS evaluates $Lasted(on, 5)$ to true whenever *on* is true in every instant among the previous 4, in which $Dist(on, -d)$ can be evaluated. Near the border, *on* can be evaluated in less than 4 instants, and thus $Lasted(on, 5)$ becomes true even if *on* does not last for at least 4 instants.

This situation is typical of every bounded operator. A possible solution consists in regarding a bounded operator as not evaluable whenever its distance from the border is less than the stated bound. In the above example, $Lasted(on, 5)$ should not be evaluated in 4, 5, 6 and 7, that is the border should be ignored. In this way the specification becomes true. However, it is possible to improve further this solution as shown next.

Consider the history depicted in Figure 7, which is similar to the history of Figure 6 but in which *on* is false at instant 5. There is no finite or infinite behavior of the specified system that may include this one, since there cannot be a timeout at instant 6. The specification should evaluate to false when interpreted

over this history. Hence, we should not ignore the border of the *Lasted* operator, but instead check if it is possible to establish its truth from the available data. Only when this is not possible, the *Lasted* operator should be regarded as not evaluable at the border.

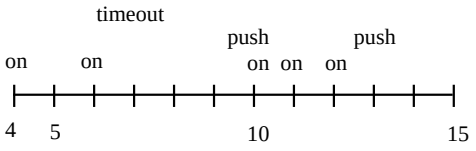

Fig. 7. An incorrect behavior of the timed lamp: there is a timeout but two instants before the lamp was off

Our proposal distinguishes the semantics of bounded operators from that of unbounded ones. Every quantification over the distance domain ΔT, defined as $1..|T|$, is assumed to be unbounded and hence treated as in MPS. Instead, in order to deal with the bounded operators, *Lasts* and the temporally symmetrical *Lasted* are added as primitive operators of the language. Their semantics is defined as follows:

- $Lasted(A, t)$ is true in i iff $\forall j, 0 < j < t, i - j \in T$ and A holds in $i - j$;
- $Lasted(A, t)$ is false in i iff $\exists j, 0 < j < t$, such that $i - j \in T$ and A does not hold in $i - j$;
- $Lasted(A, t)$ is not evaluable in i otherwise.

Symmetrically for *Lasts*.

The meaning of these clauses is that $Lasted(A, t)$ is true iff A can be evaluated and it is true in the previous $t - 1$ instants. It is false if A is false in at least one of the previous $t - 1$ instants, even if in some of these instants A cannot be evaluated. Finally, $Lasted(A, t)$ cannot be evaluated if either A cannot be evaluated in any of the previous $t - 1$ instants or A can be evaluated only in some of them and the evaluation is true. Notice that the other bounded operators of TRIO can be derived from *Lasts* and *Lasted*.

The semantics of propositional operators According to MPS if a formula A cannot be evaluated at instant i, then also $A \wedge B$ is not evaluable at i, whatever the value of B is. This semantics of propositional operators may be called *strict*: a propositional formula is evaluable only if every part of the formula is evaluable. This leads to some unpleasant drawbacks. For instance, the semantics of $Lasted(on, 5)$ is not equivalent to $Dist(on, -1) \wedge Dist(on, -2) \wedge Dist(on, -3) \wedge Dist(on, -4)$ whatever semantics we choose for the *Lasted* operator (MPS or our proposal).

Another example is the history shown in Figure 8, which should not satisfy the specification of the timed lamp example since, at instant 1, the timeout occurs and the lamp is still on.

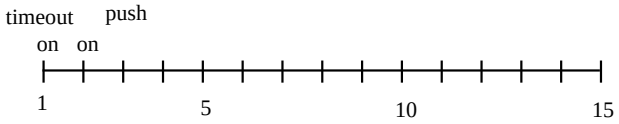

Fig. 8. An incorrect behavior of the timed lamp: there is a *timeout* and the lamp stays on

In fact, there is no finite or infinite behavior of the specified system that may include this one: the specification should evaluate to false when interpreted over this history. Instead, according to MPS the evaluation of $Alw(A1 \wedge A2 \wedge A3)$ gives the value *true*. In fact, formula (A3), $Becomes(\neg on) \leftrightarrow push \wedge Dist(on, -1) \vee timeout$, cannot be evaluated at instant 1, since $Becomes(\neg on)$ may not. As a consequence also formula $A1 \wedge A2 \wedge A3$ cannot be evaluated in 1, and therefore $Alw(A1 \wedge A2 \wedge A3)$ holds since $A1 \wedge A2 \wedge A3$ holds in every instant in which it can be evaluated (2..15 in the original MPS, 4..15 with our semantics of the bounded operators).

In order to overcome this problem, we use a different evaluation of propositional operators based on the introduction of a third truth value, called *unevaluable* (or undefined). The idea is that if A is false at instant i then $A \wedge B$ is false, even if B is not evaluable (i.e., it is not possible to establish whether B is true or false). This can be described as follows:

- $A \wedge B$ is true at an instant i iff both A and B are evaluable and true at i;
- $A \wedge B$ is false at an instant i iff either A is false (regardless of the possibility of evaluating B) or B is false (regardless of the possibility of evaluating A) at i;
- $A \wedge B$ is not evaluable at i otherwise.

This approach corresponds to adopting the Kleene's truth tables of three-valued logic [10], which are shown in Table 1. The main feature of Kleene's tables is that the value *true* or the value $false$ is returned whenever possible. In this way the previous example is now satisfactorily dealt with: $push \wedge Dist(on, -1) \vee timeout$ is true at 1, since *timeout* holds; $Becomes(\neg on)$ is defined as $\neg on \wedge Dist(on, -1)$, and therefore is false since on holds at 1. Hence, (A3) is false and therefore also $A1 \wedge A2 \wedge A3$ is false at 1. As a consequence $Alw(A1 \wedge A2 \wedge A3)$ is false.

$a \rightarrow b$				$a \leftrightarrow b$				$a \vee b$				$a \wedge b$				a `xor` b				$\neg a$	
a \ b	T	F	U	a \ b	T	F	U	a \ b	T	F	U	a \ b	T	F	U	a \ b	T	F	U	a	
T	T	F	U	T	T	F	U	T	T	T	T	T	T	F	U	T	F	T	T	T	F
F	T	T	T	F	F	T	U	F	T	F	U	F	F	F	F	F	T	F	U	F	T
U	T	U	U	U	U	U	U	U	T	U	U	U	U	F	U	U	T	U	U	U	U

Table 1. Kleene's three-valued propositional tables. U stands for unevaluable

It is easy to verify that using the new definition for bounded and propositional operators $Lasted(on, 5)$ becomes equivalent to $Dist(on, -1) \wedge Dist(on, -2) \wedge Dist(on, -3) \wedge Dist(on, -4)$, that is universal bounded quantification can be treated as an extended conjunction.

4 The formalization of the revised semantics

In this section the formalization of our proposal is presented by using a notation based on a three-valued evaluation of a formula over a finite history. Let us first define the concept of history, that is a structure on which formulas are evaluated, then the evaluation function for terms and formulas is presented.

History A *history* (or *structure*) for a formula F is a triple $S = \langle T, D, \{\Phi_i : i \in T\}\rangle$, where:

- T is the time domain.
- D is a set of interpretation domains for all identifiers occurring in F. The distance domain, $\Delta T = 1..|T|$, is an element of D. The notation $D(d)$ denotes the interpretation domain associated with identifier d.
- $\{\Phi_i : i \in T\}$ is a set of functions, providing interpretations on the domains of D for the function and predicate names of F. Φ_i provides a different interpretation for every instant i of the time domain.

Time independent functions and predicates are treated as special cases for which the different Φ_i do not change with i. If p is the name of an n-place predicate with signature $c_1, ..c_n$, Φ_i assigns an n-ary relation to it, that is $\Phi_i(p) \subseteq D(c_1) \times .. \times D(c_n)$; if f is an n-place function name with signature $c_1, ..c_n \rightarrow c_{n+1}$, then it is assigned an n-ary operation $\Phi_i(f) : D(c_1) \times .. \times D(c_n) \rightarrow D(c_{n+1})$. Time independent and time dependent constants are also assigned values by this component since they are considered as special cases of time independent and time dependent functions, respectively.

In order to interpret a formula, we need a value assignment to every variable. An *assignment* σ for a structure S is a function mapping every variable x, declared of type c_i in formula F, to a value $\sigma(x) \in D(c_i)$. A reassignment of σ for variable x is defined as any assignment σ_x that differs from σ at most in the value assigned to x. The notation S^σ represents a structure S with an assignment σ.

Evaluation of terms We define inductively a function S_i^σ which determines the value of terms and formulas for each time instant $i \in T$. The index σ, conveying the dependence of S_i from an assignment, will be omitted when no confusion can arise. S_i is defined for terms according to the following clauses.

1. $S_i^\sigma(x) = \sigma(x)$, for every (time independent) variable x.
2. $S_i(f(t_1, .., t_n)) = \Phi_i(f)(S_i(t_1), .., S_i(t_n))$, for every application of function f.
3. $S_i(c) = \Phi_i(c)$, for every constant c.

Evaluation of formulas For formulas S_i^σ returns true, false or uneval, that stands for unevaluable. The main idea is that the truth value of an atomic

formula is considered not evaluable at instant i whenever $i \notin T$: S_i in this case returns *uneval*. The value *uneval* is propagated to formulas using Kleene semantics for propositional operators, a definition equivalent to MPS for the quantifiers over the distance domain, and a new definition for the bounded operators *Lasts* and *Lasted* and bounded quantifications.

1. $S_i(p(t_1, .., t_n)) =$ **if** $i \in T$ **then** (**if** $\langle S_i(t_1), .., S_i(t_n)\rangle \in \Phi_i(p)$ **then** $true$ **else** $false$) **else** *uneval*, for a predicate p.
2. $S_i(\neg A) =$ **if** $S_i(A) = false$ **then** $true$ **elsif** $S_i(A) = true$ **then** $false$ **else** *uneval*.
3. $S_i(A \wedge B) =$ **if** ($S_i(A) = true$ and $S_i(B) = true$) **then** $true$ **elsif** ($S_i(A) = false$ or $S_i(B) = false$) **then** $false$ **else** *uneval*.
4. $S_i(Dist(A, t)) = S_{i+S_i(t)}(A)$.
5. $S_i^\sigma(\forall x A) =$ **if** $\exists \sigma_x(S_i^{\sigma_x}(A) = false)$ **then** $false$ **elsif** $\forall \sigma_x(S_i^{\sigma_x}(A) = uneval)$ **then** *uneval* **else** $true$, for a variable x of domain ΔT.
6. $S_i^\sigma(\forall x A) =$ **if** $\exists \sigma_x(S_i^{\sigma_x}(A) = false)$ **then** $false$ **elsif** $\forall \sigma_x(S_i^{\sigma_x}(A) = true)$ **then** $true$ **else** *uneval*, where x is a variable of a domain different from ΔT.
7. $S_i(Lasts(A, t)) =$ **if** $\forall j(0 < j < S_i(t) \Rightarrow S_{i+j}(A) = true)$ **then** $true$ **elsif** $\exists j(0 < j < S_i(t)$ and $S_{i+j}(A) = false)$ **then** $false$ **else** *uneval*.

Clause 4 allows the propagation of the $Dist$ operator; clause 6 is introduced to differentiate every domain different from ΔT, because ΔT is assumed to be the only unbounded domain: the other domains are bounded and are treated correspondingly. *Lasted* may be defined symmetrically as in clause 7.

5 Conclusions

A key feature of the TRIO language is its *executability*, that allows the construction of semantic tools, to help validation and verification, by means of specification simulation and test case generation. Various applications to current industrial practice [2] have used these tools.

In general, the satisfability of arbitrary first-order TRIO formulas is undecidable: a general interpretation algorithm is not guaranteed to terminate with a definite answer. To achieve executability, in TRIO the original interpretation domains, which are usually infinite, are replaced by some finite approximation thereof. Of course validating a specification using finite domains does not provide answers for the corresponding problem in the infinite domain case, but still provides useful and effective validation methods.

The definition of algorithms for the finite domain case requires a suitable finite-domain semantics, that is a semantics on finite domains that approximates the results on infinite domains. In this paper we showed how the current semantics of the TRIO language (MPS) needs to be revised in order to overcome several major problems. Our improved version of the MPS introduces a distinction between bounded and unbounded quantifiers and operators, evaluates differently the propositional operators and corrects various other flaws of

the original definitions of [7]. We believe our proposal to be an improvement also in terms of clarity and ease of comprehension.

Current work is being devoted to assert and prove approximation theorems. Satisfactory experiments have been already performed with the history checking tool, while others are currently being studied for the test case generator.

Acknowledgments
We thank Dino Mandrioli for the fruitful discussions.

References

1. R. Alur and T.A. Henzinger: Logics and Models of Real-Time: A Survey. Proc. of REX Workshop-Real-Time: Theory and Practice, Mook, The Netherlands, June 1991, LNCS 600, Springer Verlag, New York, 1992, pp. 74-106.
2. M. Basso, E. Ciapessoni, E. Crivelli, D. Mandrioli, A. Morzenti, P. San Pietro: Experimenting a Logic-based Approach to the Specification and Design of the Control System of a Pondage Power Plant. M. Wirsing (ed.), ICSE-17 Workshop on Formal Methods Application in Softw. Eng. Practice, Seattle, WA, April 1995.
3. E. Ciapessoni, E. Corsetti, A. Montanari, P. San Pietro: Embedding Time Granularity in a Logical Specification Language for Synchronous Real-Time Systems. Science of Computer Programming, 20(1993), pp. 141-171, Elsevier Publishing, Amsterdam, 1993.
4. M. Felder, A. Morzenti: Validating real-time systems by history-checking TRIO specifications. ACM TOSEM-Transactions On Software Engineering and Methodologies, vol.3, n.4, October 1994
5. C. Ghezzi, D. Mandrioli, and A. Morzenti: TRIO, a logic language for executable specifications of real-time systems. Journal of Systems and Software 12, 2 (May 1990), 107-123.
6. A. Morzenti, and P. San Pietro: Object oriented logic specification of time-critical systems. ACM TOSEM, Vol. 3, n. 1, January 1994, pp. 56-98.
7. A. Morzenti, D. Mandrioli, and C. Ghezzi: A Model Parametric Real-Time Logic. ACM Transactions on Programming Languages and Systems 14, 4 (October 1992), 521-573.
8. D. Mandrioli, S. Morasca, A. Morzenti: Generating Test Cases for Real-Time Systems from Logic Specifications. ACM Trans. On Computer Systems, Vol. 13, No. 4, November 1995. pp.365-398.
9. S. Morasca, A. Morzenti, P. San Pietro: Generating Functional Test Cases in-the-large for Time-critical Systems from Logic-based Specifications. Proc. of ISSTA 1996, ACM-SIGSOFT International Symposium on Software Testing and Analysis, Jan. 1996, San Diego, CA.
10. A. Urquhart: Many valued Logic. D. Gabbay and F, Guenthner (eds), Handbook of Philosophical Logic, Vol. III, Kluwer, London, 1986.