Providing Automated Support to Deductive Analysis of Time Critical Systems

Andrea Alborghetti, Angelo Gargantini, Angelo Morzenti
Politecnico di Milano, Dipartimento di Elettronica e Informazione
email: [garganti, morzenti]@elet.polimi.it

Abstract
We report on our experience in using a general purpose theorem prover to provide mechanical support to deductive analysis of specifications written in the TRIO temporal logic, and on applying the resulting tool to a widely known case study in the field of time- and safety-critical systems. First, we illustrate the required features for a general purpose theorem prover to satisfy our needs, we provide a rationale for our choice, and we briefly illustrate how TRIO was encoded into the prover's logic. Then we present the case study used to validate the obtained TRIO prover and to assess the overall approach. Finally we discuss the encouraging results of our experiment and provide some technical and methodological suggestions to researchers and practitioners willing to use our tool to analyze TRIO specifications, or aiming at customizing a general purpose theorem prover on any other formal language, especially if based on temporal logics.
Keywords: specification, validation, verification, time- and safety-critical systems, formal methods, temporal logic, automated theorem proving, case study, experience report.

1. Introduction

The importance of effective procedures for specification, validation, and verification in the development of correct and reliable computer-based systems can hardly be over-emphasized, especially in the case of time- and safety-critical systems. Validation and verification are most (cost) effective when performed in the initial phases of system development, before the costly phases of design and coding take place [Kem85]. This emphasizes the importance of the notation adopted for carrying out the specification phase, which must support the unambiguous description of the system requirements and at the same time allow for the (possibly automated) analysis of such specifications.

Formal methods (i.e., notations, and associated tools, having a strong mathematical foundation) have since long been considered a promising approach to address the above demands, but at the same time they where the target of many criticisms, especially coming from practitioners who were not convinced of their effectiveness in improving the quality of the developed products and of their overall convenience, notably from an economic viewpoint. In the recent past, however, some sensational failures in computer-based systems occurred (such as the Therac-25 accident, the Pentium bug, or the Arianne rocket fiasco), soon demonstrated to derive from miscarried specification and verification. They provided evidence that the high price of applying formal methods is certainly worth while for the most critical applications.

In the past years a host of formal languages and methods were introduced for the specification and analysis of critical systems, based on mathematical logic, state-transition systems, (process) algebras, etc. [H&M96] but no single language, method, or tool has gained universal acceptance nor has proved to tackle all problems in all application areas. It is now a widespread opinion that every individual notation has its own strong and weak points, and that there exist trade-offs among them.

TRIO, a language and tool set based on temporal logic, was defined and developed at Politecnico di Milano to support the specification, simulation, analysis and verification [MMG92, F&M94, FMM94] of time-critical systems. In particular, in [FMM94] we introduced an axiomatization of the logic that allows the specifier to formally derive properties of real-time systems from their specification in TRIO. We experienced however that formal verification can become difficult and error-prone when performed by hand: the likelihood of introducing errors in proofs, because of overlooked details or implicit, incorrect assumptions, can grow to the point of balancing the benefits of formal proofs. Therefore a strong need arises to provide an automated support to formal derivation. On the other hand, automated theorem proving is a highly specialized research field where very sophisticated techniques are need to obtain that particular combination of power, generality, simplicity, and flexibility needed in practical applications. For these reasons we did not consider undertaking the development from scratch of a new theorem prover for TRIO; rather, we looked for existing tools that could be employed for that purpose.

The present paper reports on our experience in using a general purpose theorem prover to provide a mechanical support to deductive analysis of TRIO specifications, and on applying the resulting tool to a widely known case study in the field of time- and safety-critical systems. In Section 2 we report a brief summary of TRIO. In Section 3 we illustrate the required features for a general purpose theorem prover to satisfy our needs, we provide a rationale for our choice of the PVS proof checker [SOR93], and we briefly illustrate how TRIO was encoded into the prover's logic. Section 4 presents the case study that we used to validate the obtained TRIO prover and to assess the overall approach. The results of our experience, discussed in Sections 5 and 6, are encouraging: proofs can be conducted almost at the same level of abstraction as in informal manual derivation, but now they are "certified" by the tool. On the contrary, one cannot expect, especially for complex proofs, that the prover "does it all by itself": our tool is in fact a proof checker, and the overall line of reasoning in a derivation (which, significantly, constitutes the most creative part) must still be provided by the user.

Altogether, we believe that our work provides evidence of the feasibility of the approach; in the present paper we supply technical and methodological information to researchers and practitioners willing to use our tool to analyze TRIO specifications, or aiming at customizing a general purpose theorem prover on any other formal language, especially if based on temporal logics.

2. TRIO: a Shortest Language Overview

TRIO is a first order logic augmented with temporal operators that allow to express properties whose truth value may change over time. The meaning of a TRIO formula

is not absolute, but is given with respect to a current time instant which is left implicit. The basic temporal operator is called Dist: for a given formula W, Dist(W, t) means that W is true at a time instant whose distance is exactly t time units from the current instant, i.e., the instant when the sentence is claimed.
Many other temporal operators can be derived from Dist. In this paper we use the following ones.

Futr(F, d)	$\stackrel{def}{=}$ $d \geq 0 \wedge Dist(F, d)$	future
Past(F, d)	$\stackrel{def}{=}$ $d \geq 0 \wedge Dist(F, -d)$	past
Lasts(F, d)	$\stackrel{def}{=}$ $\forall d'(0<d'<d \rightarrow Dist(F, d'))$	F holds over a period of length d
Lasted(F, d)	$\stackrel{def}{=}$ $\forall d'(0<d'<d \rightarrow Dist(F, -d'))$	F held over a period of length d
Until (A_1, A_2)	$\stackrel{def}{=}$ $\exists t\ (t > 0 \wedge Futr(A_2, t) \wedge Lasts(A_1, t))$	A_1 holds until A_2 becomes true
Alw(F)	$\stackrel{def}{=}$ $\forall d\ Dist(F, d)$	F always holds
AlwF(F)	$\stackrel{def}{=}$ $\forall d\ (d>0 \rightarrow Dist(F, d))$	F will always hold in the future
AlwP(F)	$\stackrel{def}{=}$ $\forall d\ (d<0 \rightarrow Dist(F, d))$	F always held in the past
SomP (A)	$\stackrel{def}{=}$ $\exists d(d<0 \wedge Dist(F, d))$	F held sometimes in the past
Som (A)	$\stackrel{def}{=}$ $\exists d\ Dist(F, d)$	Sometimes F held or will hold
UpToNow (F)	$\stackrel{def}{=}$ $\exists \delta\ (\delta > 0 \wedge Past(F, \delta) \wedge Lasted(F, \delta))$	F held for a nonzero time interval that ended at the current instant
Becomes (F)	$\stackrel{def}{=}$ $F \wedge UpToNow(\neg F)$	F holds at the current instant but it did not hold for a nonzero interval that preceded the current instant
LastTime (F,t)	$\stackrel{def}{=}$ $Past(F, t) \wedge (Lasted(\neg F, t))$	F occurred for the last time t units ago

Notice that, for the operators expressing a duration over a time interval (for example Lasts), we gave definitions where the extremes of the specified time interval are excluded, i.e. the interval is open. Operators including either one or both of the extremes can be easily derived from the basic ones we listed above. For notational convenience, we indicate inclusion or exclusion of extremes of the interval by appending to the operator's name suitable subscripts, 'i' or 'e', respectively. A few examples regarding the operators Lasts, Lasted, Until, AlwF and SomP follow.

$Lasts_{ie}$ (A, d)	$\stackrel{def}{=}$	$\forall d'(0 \leq d'<d \rightarrow Dist(F, d'))$
$Lasted_{ii}$ (A_1, d)	$\stackrel{def}{=}$	$\forall d'(0 \leq d' \leq d \rightarrow Dist(F, -d'))$
$Until_{ie}$ (A_1, A_2)	$\stackrel{def}{=}$	$\exists t\ (t > 0 \wedge Futr(A_2, t) \wedge Lasts_{ie}(A_1, t))$
$AlwF_i(F)$	$\stackrel{def}{=}$	$\forall d\ (d \geq 0 \rightarrow Dist(F, d))$
$SomP_i$ (A)	$\stackrel{def}{=}$	$\exists d(d \leq 0 \wedge Dist(F, d))$

3. Encoding TRIO into the Prover's Logic

The first choice we had to take in providing automatic support to proofs in TRIO was that of a convenient formal theory: an encoding of TRIO formulas and the desired reasoning mechanisms (inference rules) in the language of the automatic tool. In [FMM94] we introduced a Hilbert–like proof system, based on the use of *modus ponens* as the only inference rule, which is known to be well suited for studying the properties of a logic, but not for constructing readable proofs or for automatic theorem proving.

To the purpose of automation, two principal kinds of proof system are used in practice: clausal form coupled with the resolution rule [Wos84], and Gentzen–like systems [Pra65].

Resolution-based procedures find a proof by contradiction, deriving from the premises and the negation of the goal a huge number of consequences, until a contradiction is found. To improve efficiency, formulas are expressed in a very simple and rigid way, as clauses. This reduces the readability, and prevents the user from understanding the proofs or the reasons of their failure. We believe that this way it is not adequate to support validation and verification, where interaction with the user is fundamental.

Gentzen systems, instead, favor the combination of a simple interaction with the prover (to direct the proof in the more complex cases) with automated solving of simpler subgoals by means of decision procedures. Gentzen systems include a set of inference rules that naturally correspond to the meaning of every operator. For instance, the sequent

$$\frac{\Delta \vdash \Gamma, A \qquad \Delta \vdash \Gamma, B}{\Delta \vdash \Gamma, A \wedge B}$$

(where A and B are formulas, Δ and Γ are set of formulas and $\Delta \vdash \Gamma$ means that Γ is deducible from Δ) means that the formula $A \wedge B$ is deducible from a set of hypothesis, if and only if so are both A and B.

This presentation is easily understandable, which helps significantly in designing and examining proofs. Besides, these inference rules can be easily used by a prover to decompose a proof into a tree of subgoals. For instance, the rule above can be used to reduce the deduction of $A \wedge B$ to those of A and B. Furthermore, if a subgoal fails because a counterexample can be found for it, the same counterexample falsifies also the original goal.

For the above reasons, we chose a Gentzen–like axiomatization, augmenting the set of rules for the predicate calculus with those necessary for dealing with temporal aspects. We introduced some basic inference rules for modeling some basic properties of the Dist operator (a brief summary of TRIO is reported in the appendix) such as:

$$\frac{\Delta \vdash \mathrm{Dist}(A,0)}{\Delta \vdash A} \qquad \frac{\Delta \vdash \mathrm{Dist}(A,d) \qquad \Delta \vdash \mathrm{Dist}(B,d)}{\Delta \vdash \mathrm{Dist}(A \wedge B,d)}$$

and the Temporal Translation rule (TT):

$$\frac{\mathrm{Dist}(\Delta,d) \vdash \mathrm{Dist}(\Gamma,d)}{\Delta \vdash \Gamma}$$

whose intuitive meaning is that a deduction is still valid if looked from a different time instant.

For apparent reasons of cost and reliability, we aimed at encoding TRIO into a logic for which a prover already existed, and at including its proof system in the language of that prover. There exist some well known encoding methods, classified as either *syntactic* or *semantic*.

In the syntactic approach, the *source logic* is encoded into a *base logic*, the latter being used as a logical metalanguage to represent the formulas of the former, together with its inference rules, expressed through proper axioms. This approach is particularly powerful, allowing for the representation of every kind of rule, including the TT rule.

However, it complicates even the simplest proofs, for it forces the user to prove in the base logic that a sequence of steps forms a valid proof of the source logic, so that it is almost unfeasible, unless the prover itself is equipped with some mechanism that facilitates this kind of encoding. There is a wide variety of provers and logical environments, such as Isabelle [Pau90], ICLE [Daw92], Mollusc [Ric93] and others, which provide a facility of this kind, and have been used successfully for encoding in their language several first–order or higher–order logics.

Unfortunately, all these provers lack powerful decision procedures for arithmetic, which are essential when dealing with TRIO, since time is numeric in nature. The only prover we found that uses a Gentzen–like deductive system and applies powerful decision procedures for Presburger arithmetic and other decidable theories is PVS [SOR93], which we adopted for our experiment. Unfortunately, PVS provides no facility for syntactic encoding, so we looked for alternative encoding methods.

In the *semantic* approach the meaning of a formula is expressed in the base logic; this can make the proof of the encoded formula in the base logic very different from that in the source logic. In this way Hoare Logic and RTTL have been encoded within the HOL system [Gor89, CHH93], the Unity logic into the Boyer–Moore prover [Gol90] and the Duration Calculus (DC) within PVS [SS94].

A first choice of a semantic encoding of TRIO in PVS is representing TRIO formulas as functions from a temporal domain to booleans [Jef96]. This would make explicit the current time instant and we experienced that this encoding seriously jeopardizes the readability of proofs. So we looked for a suitable interface to hide the details of the encoding, as in the case of DC [SS94]. Unfortunately, the lack of the necessary documentation on the PVS prover made it difficult for us to build an interface similar to the one available for DC (one of whose constructors was also in the team of PVS developers) and, most important, prevented us from providing reliable estimates of the feasibility of the approach, especially concerning the de-coding from a formula in the internal logic back to TRIO.

For this reason we avoided the construction of an interface by using a suppressed state encoding, that considers TRIO formulas as an uninterpreted type. To provide the usual interpretation to TRIO formulas, we introduce the function *now* from the type TRIO formula to booleans. This function applied to a TRIO formula A has value true iff A is true now. This makes the overall system (PVS prover + axioms that realize the

encoding) behave exactly as a TRIO prover that applies our proof system as we described above.
To represent the desired inference rules, we introduced axioms that intuitively explain the meaning of the various operators, stating for example that *now*(A∧B) = *now*(A)∧*now*(B). Using these axioms and the inference rules of PVS, it was not difficult to implement the desired inference rules for TRIO, through the definition of suitable *strategies*, i.e. rules of the following kind:

$$\frac{\Delta \vdash \Gamma, \text{now(Dist(A,d))} \quad \Delta \vdash \Gamma, \text{now(Dist(B,d))}}{\Delta \vdash \Gamma, \text{now(Dist(A}\wedge\text{B,d))}}$$

These rules also allowed us to encode indirectly the TT rule, which does not seem to be representable directly in PVS, as the system does not provide means to add an external *Dist* operator to *all* formulas of a sequent, as required by the TT rule, in a single derivation step.

4. The Case Study

We validated our approach on the GRC (General Railroad Crossing) problem, which was recently used as a benchmark for languages and tools for critical systems analysis (for a statement of the problem, see the preface of [H&M96]). In the GRC problem several trains travel on railway tracks; a road intersects the tracks, with a bar at the crossing blocking vehicle traffic during train passage. For the sake of simplicity every train travels in the same direction: the case of trains traveling in both directions can be dealt with by symmetry. Two regions R and I, surrounding the crossing, are defined as depicted below.

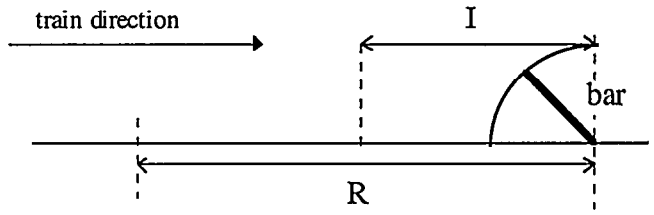

Fig. 1 The topology of the railroad crossing

Any *finite* number of trains can enter or leave any region during any finite time interval. The system must simultaneously ensure that the bar be closed whenever a train is inside region I (*safety* property), and that the bar is down only when strictly necessary (*utility* property).
It takes the train a minimum time d_m and a maximum time d_M to go from the beginning of R to the beginning of I; from h_m to h_M to go from beginning of I to its end ($d_M \geq d_m > 0$ and $h_M \geq h_m > 0$). The bar can be in two stationary positions, *open* or *closed*, or it can be moving up or moving down and is operated through commands *goUp* and *goDown*.
The following predicates formalize train movements.

RI(k)	the k-th train is entering R
II(k)	the k-th train is entering I
IO(k)	the k-th train is leaving I (and therefore R)

They denote *unique events*, i.e., they must satisfy the following axioms (expressed for event E):

$$\forall k\big(E(k) \rightarrow \big(AlwP(\neg E(k)) \wedge AlwF(\neg E(k))\big)\big)$$

i.e. E(k) is true at most in a single time instant

$$\forall k\big(E(k) \wedge k > 1 \rightarrow SomP_i(E(k-1))\big)$$

i.e. (weak) time monotonicity of parameter k

$Alw(\neg E(0))$ by convention E(0) never occurs

Informally, RI and IO represent the only input events detected by sensors; the occurrence of output events is *deduced* from the occurrence of the input ones.
Time dependent variables, CRI, CII, and CIO, count the occurrences of the corresponding events RI, II, and IO. These variables are counters, a counter is a time dependent variable, that increases for every occurrence of a given event, starting from a null initial value.

(A1) $Counter(C,E) \stackrel{def}{=}$

$$\left(\begin{array}{c} Alw\Big(\forall k\big(k \geq 1 \rightarrow \big(C = k \leftrightarrow SomP_i(E(k)) \wedge \neg SomP_i(E(k+1))\big)\big)\Big) \\ \wedge \\ Som\big(C = 0 \wedge AlwP(C = 0)\big) \end{array}\right)$$

i.e., C is a counter for event E if C is always the value of the highest k for which E(k) has occurred.

(A2) $Counter(CRI, RI) \wedge Counter(CII, II) \wedge Counter(CIO, IO)$

i.e., CRI, CII, and CIO are the counter for events RI, II, and IO, respectively.
The geometry of the region R and the behavior of the trains are formalized below.

$$\forall k\big(RI(k) \rightarrow \exists t\big(d_m \leq t \leq d_M \wedge Futr(II(k), t)\big)\big)$$

if the k-th train enters R now, then the k-th train will enter I between d_m and d_M time units in the future

$$\forall k\big(II(k) \rightarrow \exists t\big(d_m \leq t \leq d_M \wedge Past(RI(k), t)\big)\big)$$

if the k-th train enters I now, it entered R between d_m and d_M time units in the past

$$\forall k\big(II(k) \rightarrow \exists t\big(h_m \leq t \leq h_M \wedge Futr(IO(k), t)\big)\big)$$

if the k-th train enters I, then it will exit R between h_m and h_M time units in the future

$$\forall k\big(IO(k) \rightarrow \exists t\big(h_m \leq t \leq h_M \wedge Past(II(k), t)\big)\big)$$

if the k-th train exits R, then it entered I between d_m and d_M time units in the past

We assume that the bar, after a *goDown* or a *goUp* command (the two commands are mutually exclusive) reaches the final position in γ time units ($\gamma < d_m$ and velocity of motion is equal in the two directions). When the bar is moving upwards an opposite command *goDown* may be issued, causing an immediate change in movement direction.

When the bar in the closed state receives a *goUp* command, it will move upwards for γ time units or until a *goDown* is issued.

$$(M1)\ UpToNow(closed) \wedge goUp \rightarrow Until_{ie}\big(mvUp,\ goDown \vee Past(goUp, \gamma)\big)$$

A bar in the *up* position that receives a *goDown* command moves for γ and then remains closed until the next *goUp* command.

(M2)

$$UpToNow(up) \wedge goDown \rightarrow Lasts_{ie}(mvDown, \gamma) \wedge Futr\big(Until_{ie}(closed,\ goUp), \gamma\big)$$

When the bar moving up receives a *goDown* command it inverts its motion, at the same speed reaches again the closed position, and stays there until the next *goUp*.

$$(M3)\qquad \begin{pmatrix} UpToNow(mvUp) \wedge \\ goDown \wedge \\ LastTime(goUp, t) \end{pmatrix} \rightarrow \begin{pmatrix} Lasts_{ie}(mvDown, t) \wedge \\ Futr\big(Until_{ie}(closed,\ goUp), t\big) \end{pmatrix}$$

After moving up for γ time units, if there is no *goDown* command the bar stays open until the next *goDown* command.

$$(M4)\qquad Lasted_{ie}(mvUp,\ \gamma) \wedge \neg goDown \rightarrow Until_{ie}(open,\ goDown)$$

Initially, i.e., before any operation takes place, the bar is open (the bar is installed before any train arrives).

$$(M5)\qquad AlwP_i\ (\neg goDown) \rightarrow open$$

The bar control strategy computes the number of trains that are possibly in I: whenever it becomes positive a *goDown* command is issued, while whenever it becomes 0 a *goUp* command is issued. Formally, the above train number is CTPI $=_{def}$ past(CRI, d_m) - CRO (d_m in the past operator models maximum speed of trains moving from region R to region I). Let $CTPI_\gamma =_{def}$ past(CRI, d_m-γ) - CRO: $CTPI_\gamma$ account for a forward time shift of γ in issuing the command *goDown* to the bar due to the duration of the bar movement.

The commands issued to the bar are then defined as follows.

$$(C1)\qquad goDown \leftrightarrow Becomes(CTPI_\gamma > 0)$$

$$(C2)\qquad goUp \leftrightarrow Becomes(CTPI = 0)$$

These axioms must ensure then safety and utility properties, formalized as follows:

Safety: $(CII > CRO) \rightarrow$ closed

i.e., if the number of trains entered in region I is greater than those who left it, the bar is down.

Utility: $Lasted_{ii}(CII = CIO,\ \gamma) \wedge Lasts_{ii}(CII = CIO,\ \gamma + d_M - d_m) \rightarrow$ open

i.e., if the number of trains entered in region I is equal to those who left it, the bar is up (the constants γ and d_M-d_m+γ in the Utility property derive from the delay in bar rising upon train exit and from the conservative advance in bar lowering upon train enter).

5. Analysis of the Case Study

The first verification step consisted of deriving the desired *Safety* and *Utility* properties. Even with the assistance of the prover, the derivation required a great effort; moreover, to facilitate proofs, it was necessary to modify part of the specification, as it often happens in a verification activity.

As an important by-product of verification, we discovered that the strategy for governing the bar was incorrect: it allowed for the passage of a train through the crossing with the bar not closed, thus violating the *Safety* property. This constitutes a typical combination of verification and validation, whereby the requirements are assessed or found inadequate by the process of proving interesting system properties.

5.1 System Validation and Verification

The original control policy was based on the idea that any transition of CTPI from zero to a positive value should be anticipated by a similar transition of CTPIγ taking place γ time units before. Therefore, by axiom C1, when a train is inside region I the bar would be closed and *Safety* guaranteed.

We used the prover to check rigorously this idea, by formalizing the reasoning above through a sequence of lemmas and trying to prove each of them. We then discovered an error, caused by a misunderstanding of the system behavior, when we failed proving the following lemma (L7).

Alw(Becomes(CTPIγ>0)$\leftrightarrow$ Futr(Becomes(CTPI$\gamma > 0,\gamma$))

The proof attempt decomposed the original goal into four separate subgoals, some of which were derived from the specification. However we could not derive the following subgoal:

$Dist(UpToNow(CTPI = 0), \gamma)$ |– $UpToNow(CTPI\gamma = 0), Dist(CTPI = 0, \gamma)$

Then we tried to falsify it, starting from a partial model that verified the antecedent and falsified all the consequents, and then trying to complete it respecting the various specification's axioms. This activity led to a counterexample, showed in Fig. 2, that falsified lemma L7 and hence the *Safety* property.

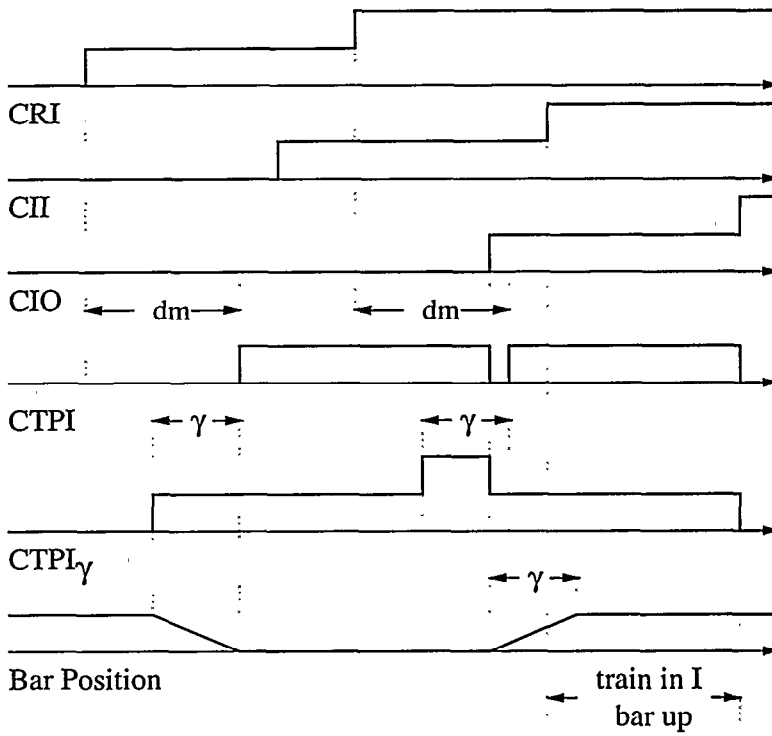

Fig. 2 A counterexample for L7 and Safety

We note that the time spent on the unfeasible proof was very short, since we quickly found the mentioned counterexample. Even more important, both the search of a proof and of a counterexample could be seen as parts of a single process, because every step of the failed proof is, at the same time, a step in the construction of the counterexample. In fact the backward application of an inference rule can be read both as: "To prove the goal I must prove all the subgoals" or "To falsify the goal I can falsify one of the subgoals".

This fact provides important methodological suggestions, for it shows that there exists a systematic way to extract useful indications from the failure of a proof, through the construction of a counterexample.

In our case, the counterexample showed clearly that the problem was originated by the fact that the increment of CTPIγ (used for sending a *goDown* command to the bar) was not necessarily from zero to a positive value, for CTPIγ could have been already positive.

The counterexample suggested also that the problem could simply be avoided by issuing the *goUp* command when CTPIγ, rather than CTPI, becomes zero. Therefore axiom C2 was modified into

$$\text{(C2)} \qquad \text{goUp} \leftrightarrow \text{Becomes}(\text{CTPI}\gamma = 0)$$

which indeed allowed us to prove both *Safety* and *Utility*.

5.2 Systematic Specifications Support Easy Proofs

Another interesting methodological issue concerns the specification of the bar. One of the key steps in proving the *Safety* property consisted of deriving the following lemma (L6):

$$\text{goDown} \wedge \text{Lasts}_{ii}(\neg\text{goUp}, t) \wedge t \geq \gamma \rightarrow \text{Futr}\big(\text{Lasts}_{ii}(\text{closed}, t-\gamma), \gamma\big)$$

Unfortunately this property, apparently depending only on the behavior of the bar, was not deducible exclusively from its specification. A simple counterexample can be constructed by considering a situation where the bar is always in the *mvDown* state and the *goDown* command has been issued periodically an infinite number of times in the past.

In practice this counterexample can be excluded based on the control policy and the definition of the counters by showing that there must be a first *goDown*. Nevertheless it showed that the specification was not well modularized. When analyzing properties of a non trivial system, modularity is a key issue for mastering complexity, and therefore, considering that the extra effort required by this deficiency could not be justified, we changed the specification of the bar, to make it more self-contained.

The need for modularity was not the only reason for changing the specification: even if we uncovered all the assumptions adopted in the first version and derived them from the specification of other system components, the particular expression of the axioms M1÷M5 would still hinder the proofs.

In fact, one of the most natural ways to conduct a proof is by analyzing the various possible cases. Of course, the case analysis must be exhaustive: in our example case, we would consider, as the various cases, the possible current system states, using an axiom like Alw(open ∨ closed ∨ mvUp ∨ mvDown) to guarantee the exhaustiveness of the analysis. Unfortunately, from the fact that the bar is in a given state, say open, the axioms do not allow one to draw directly any conclusion on other states. For instance, axiom M1 tells what happens if the bar is closed and a goUp is issued, but the case when the bar is closed and no goUp is issued is not considered explicitly. Similar argument show how difficult it would be to prove the completeness of the specification, i.e., that the desired behavior is specified for every bar state and for every issued command.

To overcome these difficulties we adopted a *state–based* specification of the bar: for each bar state we introduced an axiom describing its starting and ending conditions. For instance, the following axiom is relative to the *open* state (notice that it is structured as a set of nested implications with mutually exclusive premises).

$$\mathrm{Alw}\left[\mathrm{open} \rightarrow \left[\left(\mathrm{AlwP}(\neg \mathrm{goDown}) \rightarrow \mathrm{SomF}\left(\begin{array}{l}\mathrm{AlwP}(\mathrm{open} \wedge \neg \mathrm{goDown}) \\ \wedge\ \mathrm{goDown} \wedge \mathrm{mvDown}\end{array}\right)\right) \wedge \left(\neg \mathrm{AlwP}(\neg \mathrm{goDown}) \rightarrow \exists \mathrm{d1}, \mathrm{d2}\left[\mathrm{Past}\left[\left(\begin{array}{c}\mathrm{d2} > \mathrm{d1} \wedge \\ \mathrm{UpToNow}(\mathrm{mvUp}) \\ \wedge \\ \mathrm{Lasts}_{\mathrm{ie}}(\mathrm{open} \wedge \neg \mathrm{goDown}, \mathrm{d2}) \\ \wedge \\ \mathrm{Futr}(\mathrm{goDown} \wedge \mathrm{mvDown}, \mathrm{d2})\end{array}\right), \mathrm{d1}\right]\right]\right)\right]\right]$$

This new version of the axioms facilitated the proof of desired properties, and increased the confidence in the completeness of the specification. We do not think, however, that it would be a generally adequate solution, since it is not sufficiently readable, nor it is easy to write; moreover, it does not prevent the introduction of inconsistencies.

We then feel that a further step is needed, from purely state-based specifications to tabular specifications, as advocated by [HPSK78] and [HM83]. Applying this idea to the description of the bar, we would obtain a table with a row and a column for each state; a cell contains the condition under which the system moves from the state corresponding to its row to the state of its column. For instance, the row for the *open* state would the following,

	open	closed	mvUp	mvDown
open	¬goDown	–	–	goDown

stating that if the bar is in the *open* state and a *goDown* is issued, it goes into the *mvDown* state; otherwise it remains in the *open* state. This presentation is easy to read and write; moreover, completeness and consistency can be easily guaranteed by simple table inspections. In practice, the conditions that appear in the table can be arbitrarily complex TRIO formulas referring to the past history of the system, which gives enough power to represent more complicated behaviors, as in the case of time-outs

By means of suitable translation rules a set of TRIO axioms could be generated from the tables, to be used in proofs. In general, this could be done in a simple and convenient way in the case of discrete time domains, but not so easily when time was modeled by the set of real numbers.

5.3 Discrete vs. Dense Time Domain

This is only one of the difficulties arising from adopting the set of reals as the temporal domain. For instance we found that, in an informal proof, it was very frequent to use expressions like: "the last time in which A happened" or "the next time in which A will happen". When formalizing these concepts in PVS, it was necessary to prove, for instance, that there really existed a "last" or "next time" in which A happened. To this end, it would have been necessary, first of all, to exclude (or severely limit) the possibility of accumulation points of events, introducing a considerable amount of extra work.

As an example of this kind of complication, to prove the *Safety* property it was necessary to prove the following lemma, called Becomes_CTPIγ.

$$\mathrm{Alw}\Big(\neg\mathrm{CTPI}\gamma = 0 \rightarrow \exists t\Big(\mathrm{Past}\big(\mathrm{Becomes}(\neg\mathrm{CTPI}\gamma = 0), t\big) \wedge \mathrm{Lasted}_{ii}(\neg\mathrm{CTPI}\gamma = 0, t)\Big)\Big)$$

This lemma states that, if CTPIγ is positive, there must be a last time in which it became positive (being previously zero), and its correctness should be evident to the reader. Despite this, its proof required the introduction of many other lemmas, requiring almost a third of the total effort necessary to prove the *Safety* property. With a discrete temporal domain, instead, the proof of this lemma would have been straightforward.

Although many of the required lemmas could be exploited also in future proofs, this shows that the higher detail and precision in specifications allowed by modeling time as a continuous set does not come for free, having a negative impact on the complexity of proving even trivial facts. Therefore, one could use a discrete time model as a useful approximation for single-clock systems, keeping the full generality of real-valued time for asynchronous systems where events occur arbitrarily close in time.

5.4 Figures of Total Effort

Concerning the cost of our activity, the proof of the two properties required the introduction of 53 intermediate lemmas, reported in about 1000 pages of proof, generated by the prover (every step is the result of an interaction with the user: intermediate steps done by the prover are not reported), distributed among the various theorems and lemmas.

It is interesting to notice that lemmas concerned with properties of Counters and Events, for a total of 258 pages, are completely reusable without modifications. The percentage of possible reuse could be increased from 26% to 38%, by generalizing some lemmas to cover a broader range of cases.

The total time required for analysis was slightly more than 3 person·months, from the first serious reading of the original specification to the writing of the last page of documentation. The documentation activity took about 3 weeks and produced a 100 pages summary of the proofs.

The rest of the time was spent (i) trying to reach a sufficient understanding of the system behavior, (ii) searching for a way to formalize our reasoning, and (iii) deriving the actual proofs. After a training phase, during which we could hardly produce more than 20 pages of proofs in a day, our productivity increased significantly, reaching a rate of about 100 pages in a day in the last proofs. All the hard work was concentrated in the first two steps, witnessing the adequacy of the tool and of our encoding.

The effort required for the second phase derived mainly from the adoption of real-valued time and from the original description of the bar. In particular, we spent about a week analyzing the specification of the bar and providing an alternative formalization; then, when the alternative was found, half a day sufficed to complete the related proofs. Therefore, we are quite confident that, for adequately trained engineers, the effort should be required mainly by the first phase: understanding the

system. This does not mean that proofs would become easy and cheap: it only means that time would be spent more proficiently.

6. Conclusions and Future Developments

We summarize here a few final remarks on the experiment described in the present paper and the lessons we learned from it. Regarding the choice of a formal theory for our logic, we found that Gentzen-like systems favor human reasoning on the proofs; then we chose PVS as a tool for interactive construction of proofs, despite its lack of support to a syntactic encoding, mainly because of its powerful decision procedures.
Overall, our experience can certainly be considered successful. Indeed, the encoding and the theory could be effectively employed, in our case study, to prove system properties and to disprove false conjectures. Moreover, unsuccessful attempts to derive putative theorems led to the construction of counterexamples providing useful indications for the correction of incomplete or inconsistent specifications. This alternation between system specification, validation, and verification constitutes a very useful and effective combination of verification and simulation [M&S96].
During the analysis activity we realized that modeling time as a continuous set leads to significant increase of the complexity of proofs; this however cannon be avoided when modeling asynchronous systems.
On the other hand, the figures presented in the preceding Section 5 show that the definition and utilization of this novel approach to system analysis required a significant effort, in terms of both human and computing resources. Therefore the question arises, as it is often the case with applications of formal methods, whether the obtained results were really worth the required effort, and if this method can usefully be employed in practical, industrially-sized applications.
An impartial judgment on this crucial aspect should consider that a significant part of the effort spent in the investigation of the GRC case study derived from (self)instruction on the PVS tools (that we had never used extensively before) and from gaining experience in the use of the TRIO axiomatization and encoding for deriving system properties. In fact, even if the figures reported in Section 5 on the case study do not include the work to define the encoding of TRIO in PVS, this could be effectively validated only when applied systematically to a realistic example.
We therefore expect significant cost reductions in future applications of the proposed method, deriving from: increased knowledge of both the formal and mathematical aspects of PVS as well as of its most mundane features of the tool, which have a strong impact on its practical usability. Besides, from the development of our case study, we were able to extract some generally useful methodological guidelines. They should lead to the definition and construction of libraries of generic reusable components (PVS parametric theories) supporting the definition of high-level notions (such as states, events, counters, etc.), whose relevant features and properties would be pre-defined and proved in advance.
Even when assuming that all these improvements will be effectively realized and applied, we maintain that the analysis of complex, (time) critical systems, especially when performed by means of formal correctness proofs, is a difficult, costly activity that requires skilled, well trained personnel. In our opinion these methods can

therefore be applied with tangible advantages only to the most critical, non-standard kernel components of the developed systems. Recent advances in the technology of theorem provers, proof checkers, model checkers, and simulators, have improved the state of the art and widened their application area, but have not produced, in our view, any dramatic breakthrough.

We intend to pursue the present approach to the specification, validation, and verification of time critical systems along the following lines.

- Development of other case studies of similar size and complexity, to verify our above-reported hypothesis on diminishing costs in successive applications;
- Investigation of alternative, promising approaches to the encoding of TRIO, such as the adoption of a semantic encoding coupled with the construction of a front-end acting as a parser/unparser of the language [SS94];
- Construction of libraries of predefined theories to support reusability, modularization, and bottom-up construction of specifications and proofs;
- Integration of different, complementary tools and methods, in the same line as [Rus96], combining theorem-provers not only with model-checkers, but also with simulators/history-checkers, as advocated by [F&M94, M&S96].

Acknowledgments

We thank Ralph Jeffords for useful suggestions on the encoding of TRIO in PVS, and Dino Mandrioli for his advice on the focus and presentation of this paper.

7. References

[CHH93] R. Cardell–Oliver R. Hale and J. Herbert. "An embedding of Timed Transition Systems in HOL". *Formal Methods in System Design*, August 1993.

[Daw92] Mark Dawson, "The Imperial College Logic Environment". Technical report, imperial College of Science, Technology and Medicine, 1992.

[F&M94] 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.

[FMM94] M.Felder, D.Mandrioli, A.Morzenti, "Proving properties of real-time systems through logical specifications and Petri net models", IEEE TSE-Transactions of Software Engineering, vol.20, no.2, Feb.1994, pp.127-141.

[Gol90] D. Goldshlag, "Mechanizing Unity". In M. Broy and C.B. Jones, editors, *Programming Concepts and Methods*, North Holland, 1990.

[Gor89] M.C.J. Gordon, "Mechanizing programming logics in higher–order logic". In G. Birtwistle and P.A. Subrahmanyam, editors, *Current Trends in Hardware verification and Theorem Proving*, Springer–Verlag, New York, 1989.

[H&M96] Heitmeyer C., Mandrioli D. (editors) "Formal Methods for Real-Time Computing", John Wiley & Sons, Series Trends in Software vol. 5, 1996.

[HM83] Heitmeyer C., McLean J., Abstract requirements specifications: A new approach and its application. IEEE TSE-Transactions of Software Engineering, SE-9, 5, Sept.1983, pp.580-589

[HPSK78] Heninger K., Parnas D.L., Shore J.E., Kallander J.W., Software requirements for the A-7E aircraft. Tech. Rep. 3876, Naval Research Lab., Wash., DC, 1978

[Jef96] R.D.Jeffords, "Encoding the Real-Time Logic TRIO in PVS", Naval Research Laboratory Research Report, May 1996.

[Kem85] R.A. Kemmerer, "Testing formal specifications to detect design errors," *IEEE Transactions on Software Engineering*, vol. 11, no. 1, pp. 32-43, January 1985.

[M&S96] A.K.Mok and D.Stuart, "Simulation vs. Verification: Getting the Best of Both Worlds", Proc. of COMPASS, 11th Annual Conference on Computer Assurance, June 1996, Gaitersburg, MA.

[MMG92] A.Morzenti, D.Mandrioli, C.Ghezzi, "A Model-Parametric Real-Time Logic", ACM TOPLAS-Transactions on Programming Languages and Systems, Vol.14, n.4, October 1992 pp.521-573.

[Pau90] L. Paulson, "The next 700 theorem provers". In P. Odifreddi, editor, *Logic and Computer Science*, Academic Press, New York, 1990.

[Pra65] D.Prawitz, "Natural Deduction. A Proof Theoretical Study", Almqvist & Wiksell, Stockholm, 1965.

[Ric93] B.L. Richards, "Mollusc User's Guide". Technical report, University of Edinburgh, 1993.

[Rus96] J.Rushby, "Automated Deduction and Formal Methods", Proc. of CAV '96, Springer Verlag LNCS 1102, pp.169-183, July 1996.

[SOR93] N. Shankar S. Owre and J.M. Rushby. "User guide for the PVS specification and verification system, language and proof checker (beta release)". Computer Science Laboratory, SRI International, Menlo Park, CA 94025, USA, February 1993.

[SS94] J.U. Skakkebæk and N. Shankar, "Toward a Duration Calculus assistant in PVS", in Willem–Paul de Roever Hans Laangmaack and Jan Vytopil, editors, *Proc. 3rd Int'l Symp. on Formal Techniques in Real–Time and Fault–Tolerant Systems.* Springer–Verlag, 1994.

[Wos84] Larry Wos, Ross Overbeek, Ezing Lusk and Jim Boyle, "Automated reasoning: introduction and applications", Prentice Hall inc., 1984.