Basic Research in Computer Science

BRICS RS-94-19

Automatic Verification of Real–Timed Systems Using Epsilon

Jens Chr. Godskesen
Kim G. Larsen
Arne Skou

BRICS Report Series **RS-94-19**

ISSN 0909-0878 **June 1994**

See back inner page for a list of recent publications in the BRICS Report Series. Copies may be obtained by contacting:

**BRICS
Department of Computer Science
University of Aarhus
Ny Munkegade, building 540
DK - 8000 Aarhus C
Denmark**

**Telephone: +45 8942 3360
Telefax: +45 8942 3255
Internet: BRICS@brics.dk**

BRICS publications are in general accessible through WWW and anonymous FTP:

`http://www.brics.dk/`
`ftp ftp.brics.dk (cd pub/BRICS)`

Automatic Verification of Real–Timed Systems Using Epsilon*

Jens Chr. Godskesen Kim G. Larsen Arne Skou †

Abstract

In this paper we report on an application and extension of the theory of *Timed Modal Specifications* (TMS) and its associated verification tool Epsilon. The novel feature with which Epsilon has been extended is the ability to automatically generate *diagnostic information* in cases of erroneous refinement steps.

1 Introduction

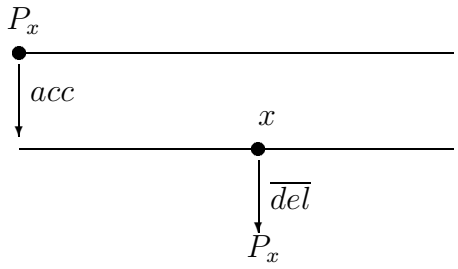
Formal techniques has since long been successfully applied for the specification and validation of concurrent and non–deterministic systems. However, most of them lack the ability of dealing explicitly and automatically with time, by means of a verification tool. So far timing properties has mostly been dealt with in a *qualitative* way: that is, only constraints on the relative ordering of a systems events are expressible. Hence, for real–time systems a possibly required delay quantity between events must necessarily be abstracted away. TMS [ČGL93] however offers the possibility to reason explicitly and *quantitatively* about time and offers together with its verification tool Epsilon the ability to perform automatic reasoning.

The theory of TMS is a real–time extension of Modal Specifications [Lar90, BL90, LT88, HL89] which in turn is an extension of process algebras, such as CCS. The real–time extension of Modal Specifications is inspired by the real–time extension of CCS by Wang presented in [Wan90].

To illustrate the design principles and the philosophy underlying the theory of TMS consider the following example. The specification of P_x below is a description of a perfect medium with a transmission delay x. Intuitively, we have that a message must be accepted at any time. This is denoted by the vertical arrow $\xrightarrow{acc}$. We adopt the convention that a labeled arrow signifies an event continuously enabled starting from the point at which the arrow is attached to a horizontal line and as long as the horizontal line continues (possible infinitely). That is, we take the horizontal lines to represent time. After acceptance of a message the medium must wait exactly x time units before delivery of the message is

*This work has been supported by BRICS, Centre of the Danish National Research Foundation, and the ESPRIT Basic Research Action 7166, CONCUR2.

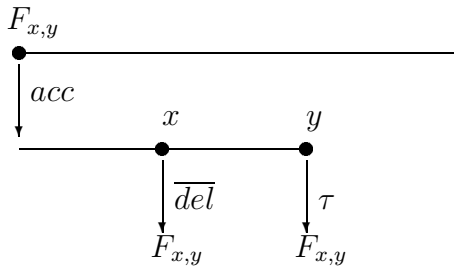
†Address: Dep. of Math. and Comp. Sc., Aalborg University, Fredrik Bajers Vej 7, 9220 Aalborg,Denmark. E–mail: {jcg,kgl,ask}@iesd.auc.dk



enabled. This is denoted in the figure by letting $\xrightarrow{\overline{del}}$ being attached precisely x time units after the point at which the arrow head of $\xrightarrow{acc}$ reaches the second horizontal line. That is, the delay x is relative to the occurrence of the acc event. Applying the syntax of TMS [ČGL93] which is an extension of Wang's TCCS [Wan90] we write

$$P_x \stackrel{def}{=} acc.\epsilon(x).\overline{del}.P_x$$

in order to express the perfect medium. Using a similar diagrammatic representation we define a faulty medium $F_{x,y}$ with transmission delay x and enforced timeout at y. Intuitively, the faulty medium behaves exactly as the perfect medium P_x except that y

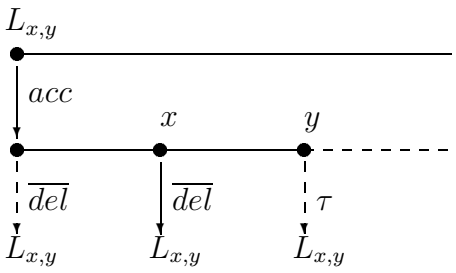


time units after acceptance the message may be lost due to a timeout. Here the internal label τ is used as the timeout event. Note, that time cannot progress beyond y. Using TCCS/TMS we may syntactically define the faulty medium by

$$F_{x,y} \stackrel{def}{=} acc.(\epsilon(x).\overline{del}.F_{x,y} + \epsilon(y).\tau.F_{x,y})$$

Suppose we have the task of defining a perfect protocol consisting of a sender S, a receiver R and a medium, say P_x or $F_{x,y}$ for some x, y. In case of P_x, given a value for x, it would be relatively easy to design S and R, simply because the message always can be read by R after some delay x. However, for the faulty medium $F_{x,y}$ the task is somewhat more complex as the design of S and R will depend on the exact values of x and y. For instance, if R is not able to engage in the delivery of the message before the medium timeouts, the sender S must necessarily retransmit the message. One would then expect the design of S and R in the case of a perfect medium P_x to be quite different from the design of S and R wrt. $F_{x,y}$.

We aim at *generality* in our design. That is, we want the theory to permit us to verify that the eventual design of S and R yields a correct protocol for a number of behaviorally different media. In particular, we would like to give a single design for S and R that would work for perfect media, like P_x, as well as faulty media, like $F_{x,y}$, for varying x and y. Actually, our goal is to design S and R with respect to a whole family of behaviorally distinct media. Assuming that S and R yield a correct protocol for a particular medium, we would expect S and R to yield correctness also for any medium which can deliver messages faster as well as for any medium which will timeout no sooner. But how to specify such families of media? The idea is to allow for *partial* or *loose* specifications which may be satisfied by several and behavioral quite different implementations. Here looseness of specifications is obtained through the introduction of two different modes of events: events which are *required* and events which are *allowed*. Diagrammatically we write required events as usual, allowed events are denoted as labels on dashed arrows. In particular we use $\xrightarrow{\tau}$ as a required timeout. Similarly, we shall write a dashed arrow with a τ label for an allowed timeout. In the case of allowed timeout we shall not prohibit the progress of time (i.e. cut a horizontal line) but rather indicate the allowance of time progression by a dashed line. Consider the (family of) media specified by



Intuitively we have, as in the previous examples, that a message must be accepted at any time. After acceptance of the message delivery is allowed immediately. However, only x time units after the acceptance the delivery is required. y time units after the acceptance a timeout is allowed to occur, but timeout is never required. In particular, time is allowed to progress so delivery can happen later than y time units after acceptance. In TMS $L_{x,y}$ may be specified as follows (with the ?–prefix constructs specifying allowed events):

$$L_{x,y} \stackrel{def}{=} acc.(\overline{del}?L_{x,y} + \epsilon(x).\overline{del}.L_{x,y} + \epsilon(y).\tau?L_{x,y})$$

The introduction of allowed and required events allow the definition of *refinement* orderings extending in a natural way the notion of process equivalences (e.g. bisimulation equivalence [Mil89, Par81], actually all of the pleasant properties of bisimulation are preserved, such as efficient decidability which is crucial for automation). Intuitively we expect a specification S to be a refinement of a specification T when all events allowed by S are also allowed by T and if all events required by T are also required by S. With this notion of refinement we expect that $L_{x,y} \lhd L_{x',y'}$ whenever $x \leq x'$ and $y \geq y'$ where we take $\lhd$ to denote refinement. Also, we expect that $P_x \lhd L_{x',y'}$ and that $F_{x,y} \lhd L_{x',y'}$ whenever $x \leq x'$ and $y \geq y'$.

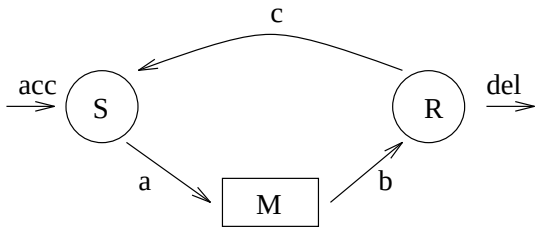
The tool Epsilon supports automatic verification of three types of refinement: strong refinement ($\lhd$), observational refinement abstracting from internal computation ($\trianglelefteq$), and refinement abstracting from time quantities ($\dot{\trianglelefteq}$). In addition, Epsilon automatically offers diagnostic information in case of erroneous design steps (a frequently occurring situation), which has proven a valuable feature in the subsequent debugging. The theoretical basis for the generation of diagnostic information is obtained through logical characterizations of the various refinements corresponding to the well known logical characterization of bisimilarity [HM85].

In analogy with TMS being a real–time extension of Modal Specifications the tool Epsilon is a real–time extension of the verification tool for Modal Specifications, Tav [GLZ89, BLS92]. The automatic refinement checking for TMS implemented in Epsilon is performed through adopting the techniques in [Čer92] and [LW90]. However, for the extension of Epsilon with the ability of generating diagnostic information it has been necessary to develop new algorithmic techniques [GL94].

Due to space–limitations this paper offers only an informal presentation. However, the work reported has shown that Epsilon together with the underlying theory TMS does indeed support the above mentioned concepts of generality in design and proofs, and in particular that supply of diagnostic information is a useful feature in case of erroneous refinements.

2 A Timed Stop–and–wait Protocol

This section reports on an analysis of a simple "stop–and–wait" protocol [Tan88, Par85] consisting of two subsystems (a sender and a receiver) interconnected by a faulty simplex medium for data and a perfect channel for acknowledgements.



We introduce the notion of a timing fault in the medium as follows: Whenever the sender has issued a message, the medium makes it available for the receiver via the b-port for a limited time period, say Y. If the receiver fails to collect the message before Y has elapsed, the medium enters a state where it (non-deterministically) *may* be lost (via a $\tau?$-transition). This is modelled as follows:

$$M(Y) \stackrel{def}{=} a.(\overline{b}.M(Y) + \epsilon(Y).\tau?M(Y))$$

As for the receiver, we want to model the fact that after having sent an acknowledgement, it takes a certain amount of time, say Z, to become ready to accept a new message

from the medium. In Epsilon this may be modelled as follows:

$$R(Z) \stackrel{def}{=} \epsilon(Z).b.del!!\overline{c}.R(Z)$$

In the above definition, we assume that the message will be delivered (via a *del*-transition) to the environment immediately. To model this, we have used the notation $a!!S$ for *urgent action transitions*, which simply abbreviates $a!!S \stackrel{def}{=} a.S + \tau.a!!S$.

Analysis With a Simple Sender

Considering the design of the sender process, we first simply ignore (naively) the fact that messages may be lost, assuming that the receiver is fast enough. The process S is based on this assumption:

$$S \stackrel{def}{=} acc.\overline{a}.c.S$$

Our timed version of the stop–and–wait protocol is then the parallel composition of the three components:

$$Protocol(Y, Z) \stackrel{def}{=} (S \mid M(Y) \mid R(Z))\backslash[a, b, c]$$

In our investigation of the protocol we have first examined the *safety* properties using the time–abstracting refinement, and thereafter we have considered the more detailed *timed* liveness properties. As for safety properties we simply want the behaviour of the protocol to be that of an infinite sequence of alternations between *acc*- and *del*-transitions, i.e. it must be an observational refinement refinement of the following specification $Spec_1$

$$Spec_1 \stackrel{def}{=} acc.del.Spec_1$$

By application of Epsilon we have examined the above requirement for various *specific* choices of the parameters Y, Z. Intuitively there are three interesting cases: $Y > Z$, $Y = Z$ and $Y < Z$ depending on whether or not the receiver becomes ready for data acceptance before the medium enters a error prone state. Taking e.g. $Y = 2$ and $Z = 1$ Epsilon confirms that indeed $Protocol(Y, Z) \stackrel{\bullet}{\trianglelefteq} Spec_1$.

However, for the instance $Y = 1$ and $Z = 2$, Epsilon returns the logical formula $[acc]\langle del\rangle tt$ as a property enjoyed by $Spec_1$ but not by $Protocol(1, 2)$. The formula indicates that there is a *acc*-transition[1] leading to a state where a *del*-transition[1] of $Protocol(1, 2)$ is not possible. So, due to our logical characterization result, the protocol is *not* in general a time–abstracted refinement of $Spec_1$ when $Y < Z$. For additional explanation a closer examination of the computation of the protocol has been carried out. This analysis revealed that there is a risk of entering a deadlocked state (i.e. a state where the only possible transitions are time progress), namely $c.S \mid M(1) \mid (\epsilon(1).b.del!!\overline{c}.R(2))$. For the case $Y = Z$ it is easily seen that the system may also deadlock for the same reason.

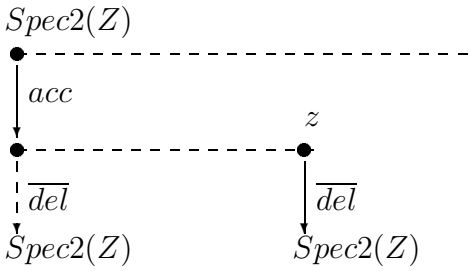
[1] with respect to a time-abstracting transition relation.

Timing Properties

Having established the safety properties of the protocol when $Y > Z$ (i.e. the receiver is fast enough to prevent the medium from timing out), we now turn to the performance characteristics in this case. Clearly the delay Z of the receiver will be a determining factor for the time delay between an input to the protocol and the succeeding output. So, one might expect the protocol to be an observational refinement of the following specification $Timedspec(A)$ which can accept an input, delay A time units, and thereafter deliver its output:

$$Timedspec(A) \stackrel{def}{=} acc.\epsilon(A).del.Timedspec(A)$$

However, in a test of e.g. $Protocol(2,1) \trianglelefteq Timedspec(1)$, `Epsilon` yields the logical property $[1/5][acc][4/5][del]ff$ enjoyed by $Timedspec(1)$ but not by $Protocol(2,1)$. This property shows that there is an implementation of the protocol which has a computation in which it can delay $\frac{1}{5}$, then receive an input, delay $\frac{4}{5}$ before delivering the output. Clearly such a computation is *not* permitted by the specification $Timedspec(1)$. In reality, the receiver–delay Z gives the *upper* limit of the input/output delay, expressed correctly by the a specification $Spec_2(Z)$ with the behaviour given below. Note that time is never



required to pass!

Our claim is now that $Protocol(Y,Z) \trianglelefteq Spec_2(Z)$ holds whenever $Y > Z$ and `Epsilon` has confirmed this for given Y, Z-values.

A Retransmitting Sender

In the general case we cannot ensure $Y > Z$, that is, we cannot guarantee the receiver to be ready for data collection before the medium will time-out. In this case ($Y \leq Z$) data may be lost by the medium, and the standard technique ([Tan88]) to handle this case is to let the sender retransmit after a certain time-out period, say X. This strategy is defined through the following sender $S_1(X)$ — introducing an additional parameter in the corresponding $Protocol(X,Y,Z)$:

$$\begin{aligned} Protocol(X,Y,Z) &\stackrel{def}{=} (S_1(X) \mid M(Y) \mid R(Z)) \backslash [a,b,c] \\ S_1(X) &\stackrel{def}{=} acc.S_2(X) \\ S_2(X) &\stackrel{def}{=} \overline{a}.(c.S_1(X) + \epsilon(X).\tau.S_2(X)) \end{aligned}$$

As before we first analyse the safety properties of the protocol — however, because of the possibility for retransmission, we only demand the specification to be included in

that of a 1–place buffer (with respect to time–abstracting language inclusion). This is expressed in the following specification:

$$Spec_3 \stackrel{def}{=} acc?del?Spec_3$$

Intuitively there are three interesting cases, i.e. $X > Z$, $X = Z$ and $X < Z$, depending on whether or not the receiver becomes ready with a frequency which is faster than the time-out period of the sender.

For the case $X > Z$, the fact that $Protocol(3,1,2) \stackrel{\bullet}{\trianglelefteq} Spec_3$ does indeed hold is confirmed by Epsilon.

Consider now the case $X = Z$ and e.g. the question $Protocol(2,1,2) \stackrel{\bullet}{\trianglelefteq} Spec_3$. In this case Epsilon returns the property $[acc][del][del]ff$ enjoyed by $Spec_3$ and not by the protocol, i.e. there is a computation of the protocol in which a message is being delivered twice. Analysing the protocol carefully, it may be seen that the receiver is able to collect a message from the medium in a state where the sender has already (incorrectly) decided (via a time-out) to retransmit the message again. This also holds for the case $X < Z$. A detailed analysis leads to the condition $X > Z$ as being sufficient for $Protocol(X,Y,Z) \stackrel{\bullet}{\trianglelefteq} Spec_3$ to hold. Again this may be confirmed by Epsilon for given values.

As for the timing analysis of the retransmitting protocol, let us now investigate if the delay parameter Z of the receiver defines the upper limit of the input/output delay. Examining whether e.g. $Protocol(3,1,2) \trianglelefteq Spec_2(2)$, Epsilon returns the property $[acc][2]\langle del\rangle tt$ as one enjoyed by $Spec_2(2)$ but not by the protocol. That is, the system may delay more than 2 time units between input/output. The reason for this is that the data may be lost in the medium *before* the time Z elapses, thereby forcing the timer X to elapse before retransmission can place. So, the time-out period X defines the upper limit of the the delay, and we may confirm this via Epsilon by proving $Protocol(X,Y,Z) \trianglelefteq Spec_2(X)$ for arbitrary parameter values satisfying $X > Z$.

References

[BL90] G. Boudol and K.G. Larsen. Graphical versus logical specifications. In *Proceedings of CAAP'90*, volume 431 of *Lecture Notes in Computer Science*, 1990.

[BLS92] A. Børjesson, K.G. Larsen, and A. Skou. Generality in design and compositional verification using tav. In *Proceedings of FORTE'92*, 1992.

[Čer92] K. Čerāns. Decidability of bisimulation equivalences for processes with parallel timers. In *Proceedings of CAV'92*, 1992.

[ČGL93] K. Čerāns, J.C. Godskesen, and K.G. Larsen. Timed modal specifications — theory and tools. In *Proceedings of CAV'93*, volume 697 of *Lecture Notes in Computer Science*. Springer Verlag, 1993.

[GL94] Jens Chr. Godskesen and Kim G. Larsen. Synthesis of distinguishing formulae for real time systems. To appear, 1994.

[GLZ89] J. Godskesen, K. Larsen, and M. Zeeberg. Tav (tools for automatic verification). users manual. Aalborg University. Denmark, 1989.

[HL89] H. Hüttel and K.G. Larsen. The use of static constructs in a modal process logic. In *Proceedings of Logic at Botik'89*, volume 363 of *Lecture Notes in Computer Science*. Springer-Verlag, 1989.

[HM85] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the Association for Computing Machinery*, pages 137–161, 1985.

[Lar90] K.G. Larsen. Modal specifications. In *Proceedings of Workshop on Automatic Verification Methods for Finite State Systems*, volume 407 of *Lecture Notes in Computer Science*, 1990.

[LT88] K. Larsen and B. Thomsen. A modal process logic. In *Proceedings LICS'88*, 1988.

[LW90] K.G. Larsen and Y. Wang. Time abstracted bisimulation: Implicit specifications and decidability. In *Proceedings of MFPS'93*, 1990.

[Mil89] Robin Milner. *Communication and Concurrency*. Series in Computer Science. Prentice–Hall International, 1989.

[Par81] D. Park. Concurrency and automata on infinite sequences. In P. Deussen, editor, *5th GI Conference*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183, 1981.

[Par85] J. Parrow. *Fairness Properties in Process Algebra*. PhD thesis, Uppsala University, Sweden, 1985.

[Tan88] A. Tanenbaum. *Computer Networks*. Englewood Cliffs, 1988.

[Wan90] Y. Wang. Real–time behaviour of asynchronous agents. In *Proceedings of CONCUR'90*, volume 458 of *Lecture Notes in Computer Science*. Springer-Verlag, 1990.

Recent Publications in the BRICS Report Series

RS-94-19 **Jens Chr. Godskesen, Kim G. Larsen, and Arne Skou.** ***Automatic Verification of Real–Timed Systems Using*** Epsilon**. June 1994, 8 pp. Appears in: Protocols, Specification, Testing and Verification PSTV '94.**

RS-94-18 **Sten Agerholm.** ***LCF Examples in HOL*****. June 1994, 16 pp. To appear in:** ***Proceedings of the 7th International Workshop on Higher Order Logic Theorem Proving and its Applications*****, LNCS, 1994.**

RS-94-17 **Allan Cheng.** ***Local Model Checking and Traces*****. June 1994, 30 pp.**

RS-94-16 **Lars Arge.** ***External-Storage Data Structures for Plane-Sweep Algorithms*****. June 1994, 37 pp.**

RS-94-15 **Mogens Nielsen and Glynn Winskel.** ***Petri Nets and Bisimulations*****. May 1994, 36 pp.**

RS-94-14 **Nils Klarlund.** ***The Limit View of Infinite Computations*****. May 1994, 16 pp. To appear in the LNCS proceedings of Concur '94, LNCS, 1994.**

RS-94-13 **Glynn Winskel.** ***Stable Bistructure Models of PCF*****. May 1994, 26 pp.** ***Preliminary draft*****. Invited lecture for MFCS '94. To appear in the proceedings of MFCS '94, LNCS, 1994.**

RS-94-12 **Glynn Winskel and Mogens Nielsen.** ***Models for Concurrency*****. May 1994, 144 pp. To appear as a chapter in the** ***Handbook of Logic and the Foundations of Computer Science*****, Oxford University Press.**

RS-94-11 **Nils Klarlund.** ***A Homomorphism Concept for ω-Regularity*****. May 1994, 16 pp.**

RS-94-10 **Jakob Jensen, Michael Jørgensen, and Nils Klarlund.** ***Monadic Second-order Logic for Parameterized Verification*****. May 1994, 14 pp.**