Verification with Real-Time COSPAN*

C. Courcoubetis[1] D. Dill[2] M. Chatzaki[1] P. Tzounakis[1]

[1]Department of Computer Science, University of Crete
and Institute of Computer Science, FORTH.
Department of Computer Science, Stanford University.

Abstract. We describe some examples using an extension of Kurshan's COSPAN system to verify bounded delay constraints in a dense time model, based on the method proposed by Dill for adding timing constraints to Büchi automata. The S/R model and COSPAN are reviewed as background, then we describe how timing can be incorporated into S/R processes, and briefly describe the modified verification algorithm. The examples consist of several time-dependent versions of the Alternating Bit Protocol and the Fiber Distributed Data Interface (FDDI).

1 Introduction

It is widely recognized that the design of protocols and concurrent algorithms is a subtle art. Human designers seem to have difficulty anticipating all the possible interactions among processes operating in parallel. The result is that many or perhaps most protocols and algorithms in use contain bugs resulting from unforseen interactions among their components. These design errors are extremely difficult to detect and debug by simulation or even by running an implementation of the system, because such systems are nondeterministic; problems are likely to be intermittent and non-repeatable.

A partial solution to this problem is to use an automatic protocol verification program to assure the correctness of the system. Such programs take advantage of the finite-state nature of many of these problems to enumerate all of the states of the system, which checking for violations of a user-provided specification. When a violation of the specification is discovered, the verifier can report an execution history that shows how the violation could occur.

Currently, one of the most sophisticated verification programs is COSPAN, developed by R. Kurshan and others at AT&T Bell Laboratories. COSPAN has been used to verify many protocols and hardware designs. COSPAN [HK89, KK86, ZH90] is based on a model which specifies the system in terms of finite-state machines coupled with a powerful communication mechanism. The model is powerful enough so that COSPAN can express and validate any finite-state property (more precisely, any property that corresponds to an ω-regular language).

* This work supported in part by the BRA ESPRIT project REACT, and by the Office of the Chief of Naval Research, Grant number N00014-91-J-1901-P00001. This publication does not necessarily reflect the position or policy of the U.S. Government.

In general, the models used by protocol verifiers abstract away from time — they model the *orderings* of events, but not their times of occurrence. However, this is not adequate for verifying protocols or algorithms that depend on timing constraints crucially for their correct operation, or that must satisfy timing requirements imposed by the external environment. As computers and networks interact more with physical devices and processes, it will become increasingly difficult to hide or ignore timing properties. But obtaining correct real-time systems is especially difficult, because timing constraints amplify the complexity of the interactions that can occur. Hence, automatic verifiers for real-time systems would be very valuable.

The nature of time is a central question that must be addressed in any system for reasoning about time. There have been many proposals for frameworks for verifying timing properties. These can be grouped into three major categories according to their underlying models of time, which we call *discrete time* [JM86, HPOG89], which considers time to be isomorphic to the integers; *fictitious clock* [AK83, AH90, Bur89, Ost90] which measures time by comparing system events with a fictitious *tick* event that occurs at regular intervals; and *dense time*, which allows an unbounded number of events to occur between two different times [AD90, Alu91, ACD90, ACD91a, ACD91b].

Each model has its adherents, and space does not permit a comparative analysis. Suffice it to say that the model used here is *dense time*, because (in the authors' opinion) it is easier to justify in terms of physical processes.

Real-time COSPAN is an extension of the COSPAN verifier to support reasoning about systems with timing constraints, where the constraints are unknown but bounded delays. A detailed description of real-time COSPAN, along with some examples, appears elsewhere [CDT]. The focus in this paper is on more significant examples that have been done with the verifier. Our extension of COSPAN uses a method similar to those proposed by Berthomieu and Menasche [BM83], time-constrained Büchi automata [Dil89], and asynchronous circuits with interval-bounded delays [Lew89]. The method uses a collection of real-valued variables that keep track of the difference between the current time and the time at which certain future events will occur. The analysis algorithm uses systems of linear inequalities to represent sets of assignments to these variables. Since there happen to be a finite number of such systems of inequalities an analysis algorithm is possible.

The basic approach is to augment the description of the system with an automatically-generated "monitor" that excludes event orderings that are inconsistent with a set of given timing constraints. The monitor works by keeping track of systems of linear inequalities. We have implemented the above ideas as an independent module *without changing* the COSPAN code. In general, we believe that the same approach can be used to splice timing analysis into other finite-state verification tools without making major changes to other parts of the system.

The paper is organized as follows. In order to make the paper self-contained we first describe the finite-state machine model and the COSPAN system. Then

we explain how timing constraints are added, and give several nontrivial example verifications of systems with timing constraints.

2 The S/R Model and the COSPAN System

2.1 The Selection/Resolution Model

The *selection/resolution* (S/R) model [AKS83, AC85, GK80, Kur90, ABM86] provides a method of describing a complex system as a set of coordinating finite state machines. S/R facilitates concise and understandable specifications by using logical predicates to describe coordination between machines. The model is described here only intuitively. More details are available in the above references.

In the S/R model, a system is decomposed into a set of simple components called *processes*; each process is an edge-labeled directed graph, (see Figure 1). The vertices of the graph are *states* of the process. Each process has a name and a *selection variable*, whose name is the process name followed by #. Each directed edge describes a state transition that is possible in one computation step; it is labeled with a *predicate* on selection variables. To make the examples easier to read, we adopt the convention that a "self-loop" transition with the label "else" is labeled with the negation of the sum of the labels of the outgoing transition.

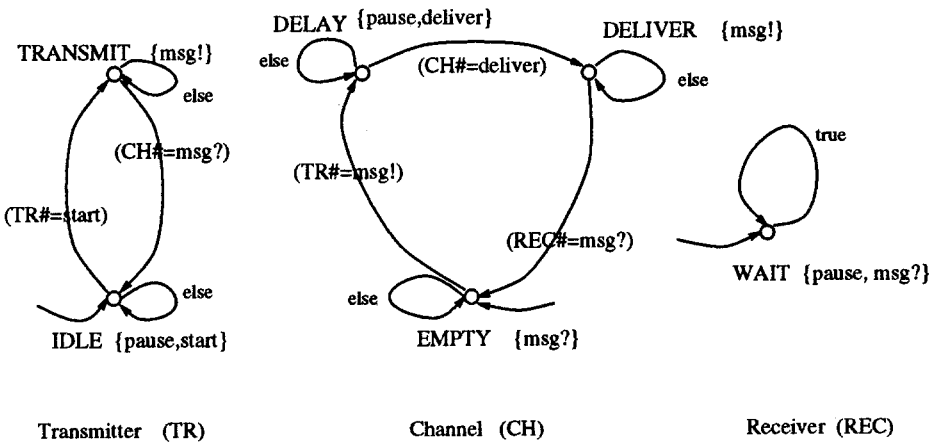


Fig. 1. A Delay Channel Example. The channel process delays the message by selecting pause.

A collection of processes characterizes a set of infinite linear histories, called *chains*. Formally, a chain is an infinite sequence of state/selection pairs $c(0) = (v(0), s(0))$, $c(1) = (v(1), s(1)), \ldots$. The chains of a collection of several processes $P_1, \ldots, P_n$ running in parallel are generated as follows. A *product state* is a vector of states of the individual processes, which summarizes the state of the entire

system. Every chain starts with the product state which consists of the initial states of the individual processes. Thereafter, each process chooses a value from among the possible selections in its current state to assign to its selection variable (this step is called *selection*). Then, each process chooses an edge label that is true for the *combined* values of all the selection variables (this step is called *resolution*). Resolution causes the processes to enter simultaneously the next states corresponding to their chosen edges.

Coordination between processes occurs because processes usually refer to the selection variables of other processes in their edge predicates. For example, in figure 1, the transmitter may select "msg!", in which case the resolution will involve the edge from state DELAY to state EMPTY in process CH.

As a convenience feature, COSPAN also allows edge labels to refer to the state variables of the processes as well as their selections.

The product just described models processes running in lock-step, so it is called the *synchronous product*. It is easy to model *asynchronous* processes by adding a selection called *pause* to each process and arranging the edge labels so that selecting pause causes a self-loop. This has the desired effect of allowing one process to take arbitrarily many steps while other processes are pausing. This feature is used in figure 1 in several places, for example to allow the transmitter TR to wait for an arbitrary amount of time in state IDLE before deciding to start transmitting.

One of the features of COSPAN is that a collection of processes can be combined into a single process with the same behavior, using the operation $\otimes$. The resulting process has states which are vectors of the individual states, selections which combine the individual selection, etc. The chains of a composite process are exactly those described above.

2.2 Monitor Processes and Proving Correctness

A *monitor* is a process whose selections are not used by other processes; it is a task that observes, but does not participate in, the execution of the system. Monitors can be used to constrain the behavior of other processes. If M does not accept all of the chains of a process P, $P \otimes M$ will contain only the chains contained by both P and M. For example, we use this feature to incorporate timing constraints into a process.

To verify correctness of a system, we need a formal specification of its desired properties. In COSPAN, these properties are called the *task*. As in other systems based on a *linear-time* models of behavior, a process satisfies a property if all of its chains satisfy the property, so the task represents the set of all desirable chains.

When verifying in COSPAN, the implementation is usually a product of processes $P = \otimes_{i=1}^{n} P_i$. Verification is especially simple if the monitor (call it TC) actually describes the *undesirable* chains[2]. Then, instead of checking that

2 COSPAN can also deal with tasks that are given in uncomplemented form. We don't use this feature here, so we will not discuss it further.

every chain of P is desirable, we can check whether *there exists* an undesirable chain; in other words, check whether the set of chains accepted by $P \otimes TC$ is empty. Emptiness can be checked in time proportional to the product of the sizes of the implementation processes and TC.

Frequently, a task is expressed as a product of several smaller tasks, $T = T_1 \otimes T_2 \otimes \cdots \otimes T_k$. In this case, we can independently check if $P \otimes TC_i$ is empty for $i = 1, \ldots, k$ for each complemented task TC_i. This is more efficient than checking for the emptiness of $P \otimes TC$, since it reduces the size of the sets of global states that must be constructed. Furthermore, constructing each TC_i is much easier than constructing TC.

2.3 Liveness Conditions

A liveness condition specifies that an event happens "eventually" without putting a specific finite bound on it. An example is a channel that is guaranteed to deliver a message eventually, but may delay arbitrary before doing so. Liveness properties cannot be specified with processes as defined above, because allowing an arbitrary finite delay necessarily entails allowing an infinite delay, also.

Liveness properties in COSPAN are handled using two additional components of a process: a finite set of *cysets* (for "cycling sets") $CY_i, i = 1, \ldots, m$, and a set of *recur edges* RE. Cysets and recur edges are used to restrict the infinite behaviors of chains: a chain c of P is *accepted* iff its state component satisfies the following conditions: (a) it does not eventually stay in some cyset CY_i, $i = 1, \ldots, m$, and (b) it does not perform infinitely often transitions from the set RE.

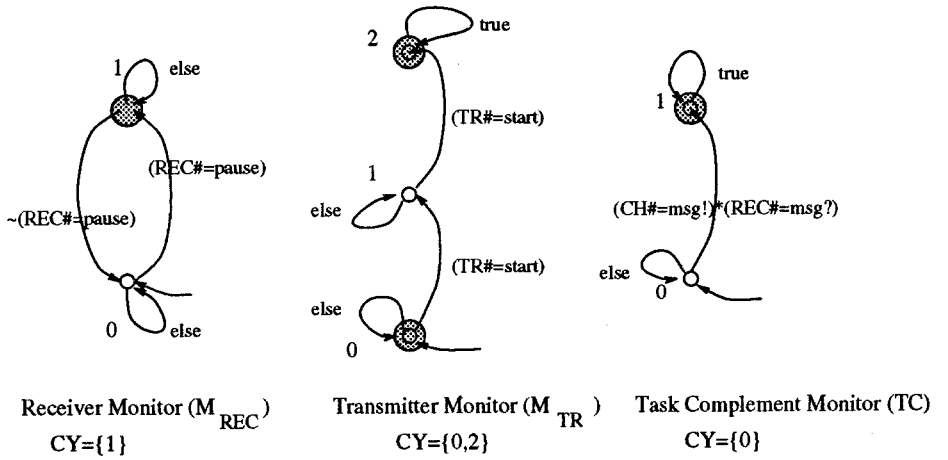


Fig. 2. The Liveness Conditions for the Specification of the Delay Channel. M_{REC} accepts all chains in which not eventually always $REC\# = pause$. M_{TR} accepts all chains in which the transmitter eventually sends a message and then stops. M_{TC} accepts all chains in which the message is never received by the receiver.

Figure 2 shows an example of how to verify some liveness properties for the delay channel example in figure 1. The specification of the system assumes that (a) the receiver always eventually checks for incoming messages, (b) the transmitter eventually transmits a unique message, and (c) the channel does not delay a message forever. These conditions are specified using the monitors M_{REC}, M_{TR} in Figure 2, and by associating the cyset {DELAY} with the channel process CH (we follow the convention of eliminating one set of braces when CY contains only a single set). The task is that the message will be eventually received. This is encoded by the task complement monitor M_{TC}.

3 The Timing Extension

The approach we use to incorporate timing into COSPAN adds to the system description a set of monitor processes which accept only the chains that are consistent with the specified timing constraints.

The specifications of a *timed process* consist of two parts: the *time-independent part* and the *time-dependent part.* The time-independent part specifies a superset of the possible chains that can occur in the actual system, which includes some chains that are not consistent with the timing constraints.

The time-dependent part is represented by a set of constraints of the form "if $c(i)$ satisfies property Q and $c(j)$, $i < j$, satisfies property R, then $l \leq t(j) - t(i) \leq u$", where $c(i)$ and $c(j)$ are elements of the chain of the timing-independent process and l and u are constants. For example, to model a traffic light for which the duration of the green light is between 2 and 3 seconds, Q and R can be defined respectively as "in $c(i)$ the light turns green", "$c(j)$ is the first chain state after $c(i)$ in which the light is red", we get a timing constraint of the form mentioned above with $l = 2, u = 3$.

Each timing constraint is encoded by a *logical timer* T. T has a *set condition* which is property Q, an *expire condition* which is property R, and an *interval specification* which (in the above example) corresponds to the interval $[2, 3]$. The interpretation is that when T is set, it will expire at some arbitrary time in the interval $[2, 3]$ from the time it was set. If needed, we associate with a logical timer a *cancel condition*; when such condition is satisfied after the timer is set, it deactivates the timer (no expiration will take place in the future).

The semantics we give to chains is the following. We assume that a chain corresponds to the sampling of the history of the real-time system at the transition epochs (i.e., immediately after a transition). We also allow an arbitrary number of transitions to occur in zero time.

This time-dependent part of the specification is translated automatically by our extension of the original COSPAN software into a monitor process M_T by using the approach in [Dil89]. This monitor does not accept the chains of the untimed part of the specification which violate the real-time constraints. One can prove by using the results in [Dil89] that the set of timing-consistent chains is the intersection of the timing-independent and timing-dependent processes, in other words, the chains of $P \otimes M_T$. A timed process has no accepted timed

behavior (real-time history) iff there is no timing-consistent chain in $P \otimes M_T$, hence iff $P \otimes M_T$ is the empty process.

The monitor works by inferring timing information for the history of timer events in the chains. It stores the timing information in its states and uses it to decide which timing events are allowed next. If a transition on a particular set of timing events is allowed, the monitor updates the state to reflect its inferred changes in timing state. If the events are not allowed, the monitor enters and remains in a "dead" state in which the chain is not accepted (a cyset).

For example, in the case of the traffic light, while the snapshots of the system satisfy the property that the traffic light is green, the time can not progress by more than three seconds. This excludes the occurrence of some other events before the light turns red, if the timing information inferred from the occurrence of these events contradicts the above information.

M_T not only keeps track of the time, but also enforces some timing-independent properties of timers: A timer cannot expire unless it is set, and a timer that is set will eventually expire (a liveness condition).

A product of timed processes can be defined by doing the usual product operation on the untimed parts (to give a new untimed part) and taking the *union* of the timers, to give the set of timers of the product.

Once M_T has been constructed and added to the system description, time-independent properties, which specify only orderings of system events but not time of occurrence, can be verified immediately, using COSPAN as described in the previous section.

A more interesting problem is to verify timed properties, for example, "the time between any two consecutive receptions of a message by the receiver is always less than 2 seconds", or "no two interrupts are sent within less than 1 ms".

Complementing timed processes is very difficult, so we finesse this issue by providing the task complement *directly* by specifying undesirable timed behavior in terms of some timed process, and then checking the emptiness of the composition of the timed processes corresponding to the specification of the system and the task complement. This approach is valid since if there is some "bad" timed behavior, it must satisfy both the timed specification of the system and the timed specification of the complement of the task. Of course our approach is applicable only when the task complement can be expressed as a timed process. In most cases generating a timed task complement is straightforward.

4 Examples

In this section we present S/R specifications and verification results for two well known communication protocols in which real-time is important. Timing information has been included in the S/R models for the Fiber Distributed Data Interface (FDDI) and the Alternating Bit (ABP) Protocols. There is insufficient space to include the full COSPAN specifications, so we summarize here the time related modeling details and the verification objectives.

4.1 FDDI Communication Protocol

Verification of the timing properties of the FDDI timed token access protocol is a challenging task. Our effort is to model FDDI and prove the timing requirements of the protocol with COSPAN. A brief description of FDDI is given below.

A station that wishes to transmit waits until a token frame is released by the previous station on the ring. After seizing the token the station may transmit frames until it has no more data to send or until a token-holding timer expires. This timer was set at the previous time at which the station received a token, and expires after some constant amount of time (which is the same for all the stations on the ring) denoted by TTRT. (This corresponds to the operation of the station for sending "asynchronous" traffic. The FDDI standard is more complex and allows the transmission of "synchronous" traffic as well. A station can always send synchronous traffic at the expense of the asynchronous one. We do not model this for reasons of simplicity, and because to our knowledge, all current FDDI implementations do not support synchronous traffic.) An important property of this protocol is (a) *fairness*, i.e., each station always eventually gets the opportunity to transmit for some positive amount of time and (b) *bounded medium access time*, i.e., the time needed for a station to access the medium is bounded above by some constant, uniformly for all the stations. We have used our method to prove that FDDI satisfies the above properties for a large choice of parameters. Of course our method can not verify the correctness of the protocol in the general case, since we cannot verify parametric specifications.

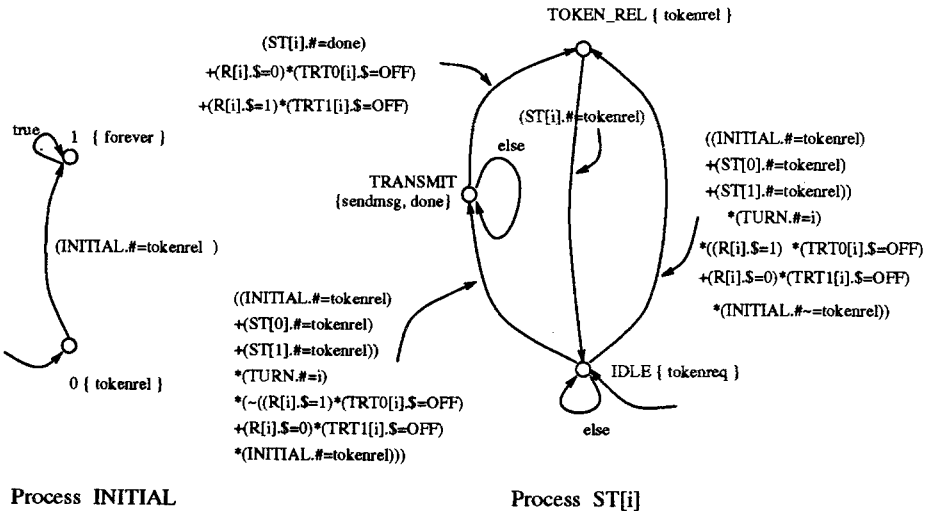
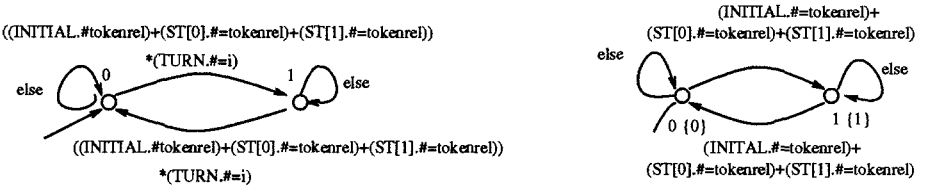


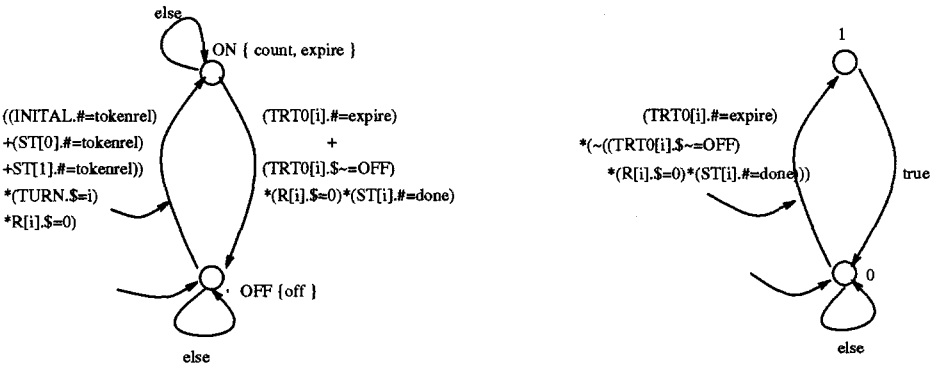
Fig. 3. S/R specification of the initialization process and stations' process.

We have modeled the protocol as follows. The processes of the specification are described in figures 3, 4, 5, 6, 7, and 8. A process called ST is used to model



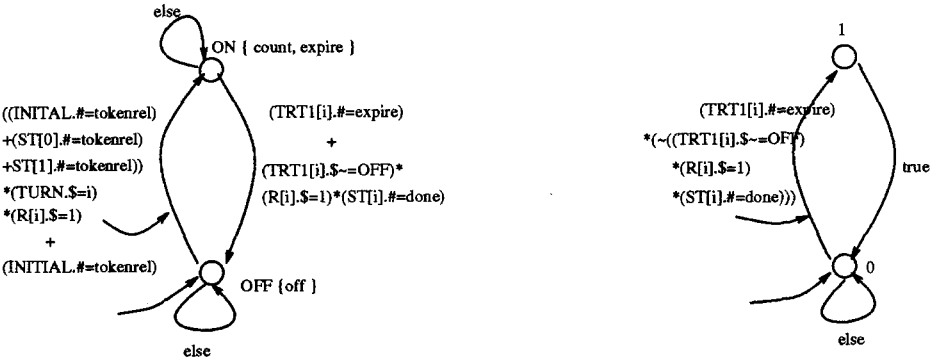
Process R[i] Process TURN

Fig. 4. R[i] keeps track of the timer that is going to be set with the next token arrival at station i. TURN keeps track of the station that will have the token next time.



Process TRT0[i] Monitor MTRT0[i]

Fig. 5. Actual timer process TRT0[i]. MTRT0[i] is a monitor that provides the information about the time TRT0[i] switches state to OFF.



Process TRT1[i] Monitor MTRT1[i]

Fig. 6. Actual timer process TRT1[i]. MTRT1[i] is a monitor that provides the information about the time TRT1[i] switches state to OFF.

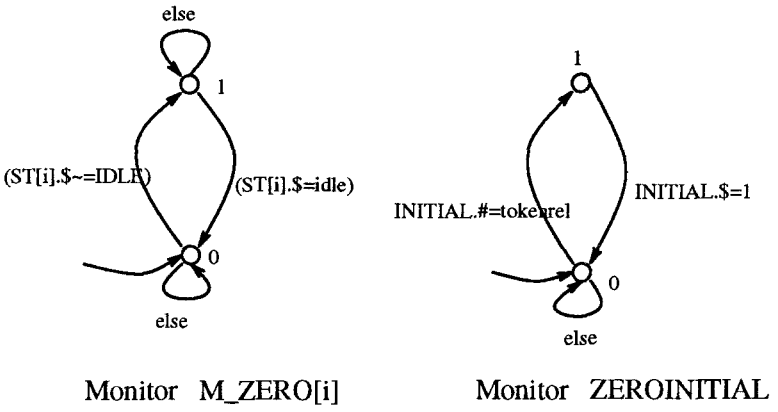


Fig. 7. Monitor process M_ZERO[i] that provides the information about the first time ST[i] goes to IDLE state. Monitor process ZEROINITIAL that provides the information about the first time INITIAL goes to state 1.

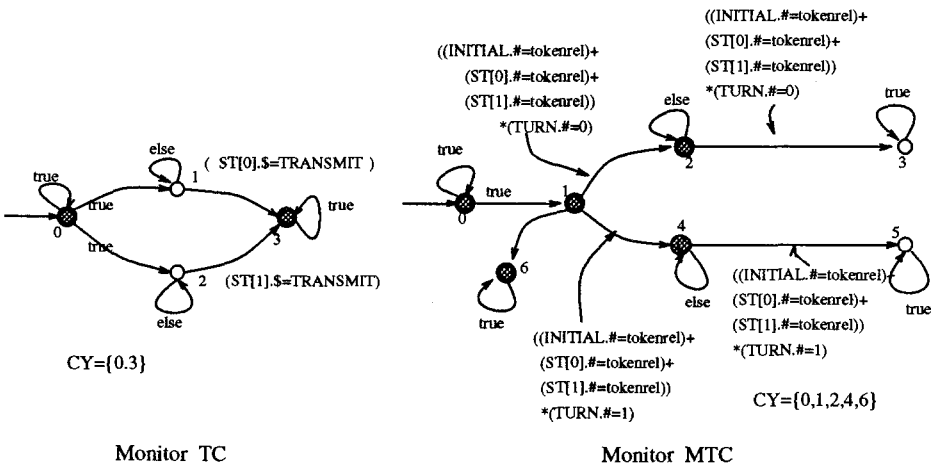


Fig. 8. Task Complement. TC accepts all chains in which some station will transmit finitely many times. MTC is . monitor which tags two consecutive token reception by a station.

stations on the ring. A station can be either in an idle state where it requests for the token, in a transmitting state where it can transmit frames, or in a token-releasing state where it passes the token to the next station. A process called TURN keeps track of which station is going to receive the token next time (for modeling the ring topology). The most interesting part of our specification is the modeling of the timers which are a combination of actual and logical timers. This combination is explained below. Each station has two actual timers which are modeled by the processes TRT0, TRT1. Whenever a token arrives to station i, either TRT0 or TRT1 is set, depending on which state process R is in; R keeps

track on the timer that is going to be set by the next token arrival, and there is one such process per station. In our specification TRT0 (TRT1) is set at every odd (even) numbered token arrival at the station. For initialization, TRT1 is set at time 0 for all stations in the ring.

After receiving the token for the k-th time, a station has permission to transmit until the relevant timer expires: if k is odd (even) the relevant timer is TRT1 (TRT0). If this timer has already expired, when the token arrives, the station must immediately release the token without going through the TRANSMIT state. Attached to each actual timer is a logical timer called L_TR_TIMER0, L_TR_TIMER1 respectively, which is set whenever the respective actual timer is set, expires when the respective timer expires, and counts exactly TTRT time units.

An other assumption in our model is that our ring has negligible propagation delays (small with respect to TTRT; this is a valid assumption for most practical cases). We model this by associating logical timers of zero duration with the transitions modeling the exchange of the token by two stations. We also add such a logical timer to model the fact that when the corresponding timer has expired and the station must release the token, this occurs within zero time (fast hardware).

We prove the two tasks mentioned above by using the task–complement monitor processes TC and MTC respectively. The first, TC, guarantees that our protocol is fair, which means that always eventually all the stations have the opportunity to transmit frames (i.e., go to the TRANSMIT state), while the second is a timed task which verifies that 2*TTRT is an upper bound of the token rotation time, that is between any two consecutive token receptions by a station. TC accepts all histories in which for some station state TRANSMIT appears finitely many times. MTC "tags" nondeterministically two consecutive token receptions by some station. We use the logical timers L_TIMER_MTC (one per station) with interval specification $(2 * TTRT, \infty]$ to count the time between the above events. Hence MTC accepts all histories for which there is a station which receives the token in two consecutive times in more than $2 * TTRT$ time units apart.

The reader will notice that in order to model set and expire conditions for logical timers of the form "process X enters state Y", we need to add monitors which capture the first such time in the chain that process X finds itself in state Y, for each different sojourn of the process in this state. As a final remark, for simplicity our specification is given in terms of a ring consisting of two station. This can be easily upgraded to an arbitrary number of stations.

4.2 Alternating–Bit Communication Protocol

Our model of the alternating–bit protocol is essentially based on the one presented in [ACW90]. Upper layers of this protocol are modeled by two processes called S and R, that generate and receive messages respectively. Two groups of processes model the transmitting and receiving part of the protocol in a peer to peer level. Finally, the lower layers constisting of two half–duplex communi-

cation channels that may lose messages and acknowledgements, are modeled by two corresponding channel processes.

Several logical timers have been introduced in order to add timing constraints. These include:

1. The resetable logical timer TICLK. This logical timer is used to supply the real-time information to the retransmission timer TI of the protocol. Its set condition is the same as the set condition of TI, and its expire condition is the expiration of TI, i.e. (TI.# = to). Its interval specification is $[TIME, TIME]$, which implies that the expiration event will occur exactly $TIME$ time units after the set event. Finally, if the correct acknowledgement is received before TI expired (and hence before TICLK expired), then TICLK is canceled.
2. The logical timer CHO_TIMER. This logical timer models the transmission delay of the outgoing channel where messages are sent. CHO_TIMER is set whenever the outgoing channel is handed a message and expires when the outgoing channel delivers or looses the message.
3. The logical timer CHI_TIMER. This logical timer models the transmission delay of the incoming channel where acknowledgements are sent. It is set when the incoming channel is handed an acknowledgement and expires when this acknowledgement is delivered or lost.

These three timers suffice to proof correctness of the protocol, e.g. in-order message delivery. The task complement used in this case is TC as in [ACW90]. However to prove that a message is delivered within specific time bounds, we have to use another logical timer DELIV_TIME, and a new task complement monitor DELIVERY_TIME. This timer is set when the sender decides to send a tagged message, and expires when the message reaches the receiver. Another property to check is whether duplicate copies of a message are ever sent through the outgoing channel under the assumption that the incoming channel does not loose acknowledgements. We have showed that presence of duplicate messages depends on the relation of the timeout value TIME to the sum of the communication channels delays. To prove this we added a new logical timer ZERORECDELAY which ensures that the receiving protocol processes messages in zero time, and we used the monitor DUPLICATES for the task complement. This monitor goes to an error state when a duplicate of a message arrives at the receiving end.

Next we used an extension of the ABP to include the upper protocol layers, in order to study buffer occupancy at the interface above ABP. We added an external message producer PROD, an external message buffer BUF, and we have changed sender S to send messages that PROD generates (this version of the sender process is in the appendix). In this model a message coming from PROD is either temporarily held in BUF in case communication channels are busy, or is directly handed to the transmission protocol using a cut-through mechanism. An logical timer PROD_TIMER controls the rate at which PROD generates messages and a new task complement BUFTHRESHOLD checks whether the buffer occupancy ever exceeds a certain threshold under the assumption that no messages or acknowledgements are lost (this version of the communication

channels are in the appendix). Clearly, this depends on the rate at which the producer process generates messages and on the delay characteristics of the channels. We have checked the buffer occupancy for a large range of system parameters.

References

[ABM86] S. Aggarwal, D. Barbara, and K. Z. Meth. Spanner - a tool for the specification, analysis, and evaluation of protocols. *IEEE Trans. on Software Engineering*, 1986.

[AC85] S. Aggarwal and C. Courcoubetis. Distributed implementation of a model of communication and computation. In *Proceedings of the 18th Hawaii Intl. Conference on System Sciences*, pages 206–218, January 1985.

[ACD90] R. Alur, C. Courcoubetis, and D. Dill. Model-checking for real-time systems. In *Proceedings of the 5th Symposium on Logic in Computer Science*, pages 414–425, Philadelphia, June 1990.

[ACD91a] R. Alur, C. Courcoubetis, and D. Dill. Model-checking for probabilistic real-time systems. In *Proceedings of the 18th ICALP*, pages 115–126, Madrid, July 1991.

[ACD91b] R. Alur, C. Courcoubetis, and D. Dill. Verifying automata specifications of probabilistic real-time systems. In *Proceedings of the REX Workshop*, Plasmolen, June 1991.

[ACW90] S. Aggarwal, C. Courcoubetis, and P. Wolper. Adding liveness properties to coupled finite-state machines. *ACM Transactions on Programming Languages and Systems*, 12(2):303–339, 1990.

[AD90] Rajeev Alur and David Dill. Automata for Modeling Real-Time Systems. In *Automata, Languages and Programming : 17th Annual Colloquium*, volume 443 of *Lecture Notes in Computer Science*, pages 322–335, 1990. Warwick University, July 16-20.

[AH90] R. Alur and T. Henzinger. Real-time logics: complexity and expressiveness. In *Proceedings of the 5th Symposium on Logic in Computer Science*, Philadelphia, June 1990.

[AK83] S. Aggarwal and R.P. Kurshan. Modelling elapsed time in protocol specification. In H. Rudin and C.H. West, editors, *Protocol Specification, Testing and Verification, III*, pages 51–62. Elsevier Science Publisers B.V., 1983.

[AKS83] S. Aggarwal, R. P. Kurshan, and K. K. Sabnani. A calculus for protocol specification and validation. In *Protocol Specification, Testing and Verification, III*. North-Holland, 1983.

[Alu91] Rajeev Alur. Techniques for automatic verification of real-time systems. Technical Report STAN-CS-91-1378, Department of Computer Science, Stanford University, August 1991. Ph.D. Thesis.

[BM83] B. Berthomieu and M. Menasche. An enumerative approach for analyzing time petri nets. In *Information Processing*, pages 41–46. Elsevier Scinece Publishers B.V. (North-Holland), 1983.

[Bur89] J. R. Burch. Combining CTL, Trace Theory, and Timing Models. In J. Sifakis, editor, *Automatic Verification Methods for Finite State Systems*, volume 407 of *Lecture Notes in Computer Science*, pages 334–348. Springer–Verlag, 1989.

[CDT] Costas Courcoubetis, David L. Dill, and Panagiotis Tzounakis. Adding Dense Time Properties to Finite-State Machines: the Tool Cospan. Submitted to a journal.

[Dil89] D. Dill. Timing assumptions and verification of finite-state concurrent systems. In *Proc. Workshop on Computer Aided Verification*, Grenoble, June 1989. Lecture Notes in Computer Science, Springer-Verlag.

[GK80] B. Gopinath and B. Kurshan. The selection/resolution model for coordinating concurrent processes. In *AT&T Bell Laboratories Technical Report*, 1980.

[HK89] Z. Har'El and R. Kurshan. Automatic verification of coordinating systems. In *Proceedings of Workshop on Automatic Verification Methods for Finite-State Systems*, Grenoble, June 1989. Springer Verlag.

[HPOG89] N. Halbwachs, D. Pilaud, F. Ouabodessalam, and A-C. Glory. Specifying, programming and verifying real-time systems using a synchronous declarative language. In J. Sifakis, editor, *Automatic Verification Methods for Finite State Systems*, volume 407 of *Lecture Notes in Computer Science*. Springer–Verlag, 1989.

[JM86] F. Jahanian and A. K-L. Mok. Safety analysis of timing properties in real-time systems. *IEEE Transactions on Software Engineering*, 12(9), September 1986.

[KK86] J. Katzenelson and B. Kurshan. S/R: A language for specifying protocols and other coordinating processes. In *Proc. 5th Ann. Int'l Phoenix Conf. Comput. Commun., IEEE*, 1986.

[Kur90] R. Kurshan. Analysis of discrete event coordination. *Lecture Notes in Computer Science*, 480, 1990.

[Lew89] H. R. Lewis. Finite-state analysis of asynchronous circuits with bounded temporal uncertainty. Technical Report TR-15-89, Aiken Computation Laboratory, Harvard University, July 1989.

[Ost90] J. Ostroff. *Temporal Logic of Real-Time Systems*. Research Studies Press, 1990.

[ZH90] R. P Kurshan Z. Har'El. Software for analytical development of communication protocols. *AT&T Technical Journal*, 1990.