Model Checking of Real-Time Systems: A Telecommunications Application

Rajeev Alur, Lalita Jategaonkar Jagadeesan, Joseph J. Kott, and James E. Von Olnhausen

May 1997

To appear in the Proceedings of the International Conference on Software Engineering, May 1997

Model-Checking of Real-Time Systems: A Telecommunications Application

Experience Report

(To appear in the Proceedings of the International Conference on Software Engineering, May 1997)

Rajeev Alur[*], Lalita Jategaonkar Jagadeesan[†], Joseph J. Kott[‡], and James E. Von Olnhausen[‡]

ABSTRACT

We describe the application of model checking tools to analyze a real-time software challenge in the design of Lucent Technologies' 5ESS telephone switching system. We use two tools: COSPAN for checking real-time properties, and TPWB for checking probabilistic specifications. We report on the feedback given by the tools, and based on our experience, discuss the advantages and the limitations of the approach used.

Keywords

Telecommunications software, Formal methods, Model checking, Real-time verification, Probabilistic verification.

INTRODUCTION

The design of concurrent real-time systems is a notoriously difficult problem. In particular, the interaction of concurrent processes in large real-time systems often leads to subtle bugs that are extremely difficult to discover using the conventional techniques of simulation and testing. Automated verification based on model-checking promises a more effective way of discovering design errors. In this approach, a high-level model of the system under design is described in a formal notation, and the verifier automatically checks whether the given model satisfies correctness properties such as absence of deadlocks or unreachability of "bad" states. This check is performed, unlike simulation, by exploring all possible interactions of the concurrent components. If the property is violated, the tool reports a scenario as evidence of the violation. In recent years, there has been steady progress in the capabilities of tools for automated verification. Current tools incorporate a variety of algorithmic heuristics, and case-studies, usually in the hardware domain, suggest that these tools can handle real-world applications (see [8] for an introduction to model checking).

Lucent Technologies' 5ESS®telephone switching system is a concurrent real-time system comprised of several million lines of C code. The 5ESS software consists of thousands of interacting concurrent processes, all with stringent real-time constraints. One component of the 5ESS software is responsible for detecting malfunctions on the hardware connections between switches. In a recent 5ESS software release, the hardware facilities were substantially upgraded, and consequently the possible rate of incoming alarms increased dramatically. It was discovered during testing that, under certain conditions in which the incoming alarm rate was high, the design violated some correctness conditions such as absence of process abortion and buffer overflow. After much simulation, the software was redesigned to perform its computations significantly faster by reducing the number of required database operations. A battery of tests seemed to indicate that the redesigned software prevents the undesirable conditions. As part of a collaboration between research and development, we are studying the application of model checking technology to this 5ESS software.

Since the correctness of the design crucially depends on the real-time constraints, we need a model checker that supports real-time verification. One such verifier, developed at Bell Laboratories, is called COSPAN [3]. We constructed a high-level model of the relevant functional and real-time aspects of the design, and used COSPAN to detect scenarios leading to conditions like queue overflow or process abortion. As expected, the model of the earlier version of the 5ESS software violates these properties. However, to our surprise, we also discovered that in the model of the new version of the 5ESS software, the queue may still overflow under certain conditions

[*] Computing Sciences Research Center, Bell Laboratories, and Department of Electrical Engineering & Computer Science, University of California, Berkeley, CA 94706 (USA), alur@ic.eecs.berkeley.edu

[†] Software Production Research Department, Bell Laboratories, 1000 E. Warrenville Rd., Naperville, IL 60566 (USA), lalita@bell-labs.com

[‡] Network Systems Platform Development, Lucent Technologies, 1200 E. Warrenville Rd., Naperville, IL 60566 (USA) jjkott@lucent.com, n9uzc@lucent.com

when the incoming rate is high. COSPAN has generated scenarios under which this problem may arise, and we are currently running these scenarios as test cases in the 5ESS lab environment.

To verify the intuition that the redesigned software was an improvement over the old one, we also constructed high-level probabilistic models of the two versions. The use of probabilities, while not supported by COSPAN, seems useful in capturing scheduling assumptions more realistically. The verification of models with real-time and probabilities is supported by a tool called TPWB [11, 13]. While computational requirements forced us to significantly scale down many of the parameters (such as the number of incoming alarms), we still could obtain useful feedback from the tool.

In addition to generating potentially useful test cases, we believe that the use of model checking aids in the design and redesign of real-time software, since the model can be easily modified to reflect changes in the functional and real-time requirements. For example, one can verify whether a decrease in the potential incoming alarm rate or a speedup in computation would avoid these queue and timing difficulties.

The paper is organized as follows. An overview of the 5ESS software and this application is given in the following section. The next section describes the COSPAN verifier, our COSPAN model and the analysis; the following section describes the TPWB tool, our TPWB model and the analysis. Our observations, along with the limitations of these approaches, are reported in the last two sections.

THE 5ESS SWITCHING SYSTEM

The 5ESS Software

Lucent Technologies' 5ESS telephone switching system [20] is a highly reliable, concurrent real-time system which provides telecommunications services, including the connection of telephone calls.

The 5ESS software is comprised of several million lines of C code, and consists of thousands of interacting concurrent processes residing on numerous processors. Each process has a fixed priority; scheduling is non-preemptive and respects process priority. In particular, on each processor, the operating system indicates to the scheduler when processes are ready to be scheduled – for example, when messages are sent to a process or when a timeout expires. The scheduler then passes control to a process in the highest-priority class ready to be scheduled: if more than one such process exists in this class, the choice is made according to the order in which the processes gave up control. The scheduled process then voluntarily indicates to the scheduler when it is ready to relinquish control, and the cycle is repeated.

5ESS processes all have stringent real-time constraints. For example, every process, when scheduled, must voluntarily relinquish control within a bounded amount of time. There are two types of time-slices: a *recommended time-slice* in which design guidelines suggest that a process voluntarily relinquish control, and a longer *maximum allocated time-slice*. If a process exceeds its maximum allocated time slice, it is killed by the operating system; clearly an undesirable consequence. In addition, most processes have real-time requirements on the rate at which they process their inputs. Together, these requirements contribute to the overall reliability of the 5ESS switching system.

Carrier Group Alarms

In switches, long-distance telephone calls are typically routed through a network of hardware, referred to as *carrier groups*. Status changes on carrier groups – such as malfunctions or recoveries from malfunctions – appear as inputs to the switch. In response to such inputs, the switch removes/restores the associated carrier groups and routes new calls over functioning carrier groups.

One component of the 5ESS software is responsible for handling alarms that signal malfunctions on carrier groups [14]. This component consists of several interacting concurrent processes divided among the *5ESS Peripheral Control* software and the *5ESS Carrier Group Alarms* software. In particular, at fixed intervals, a high-priority process in the Peripheral Control software samples the carrier group alarms and places them in a bounded-size queue. A lower-priority process in the Peripheral Control software removes alarms from this queue and calls a Carrier Group Alarms software function to perform database lookups and to send information about each alarm to a Carrier Group Alarms software process. Clearly, in order to avoid the loss of alarm information, the queue must not overflow. As described above, the design should also guarantee that no process ever exceeds its time-slice.

A Real-Time Difficulty

In a recent 5ESS software release, the hardware facilities were substantially upgraded, and consequently the possible rate of incoming carrier group alarms increased dramatically. It was discovered during testing that, under certain conditions in which the incoming alarm rate was high, the alarm queue would sometimes overflow and the lower-priority Peripheral Control process would sometimes exceed its time-slice and be killed. In the context of the new software release, these difficulties arose from the subtle interactions between the Peripheral Control software and the Carrier Group Alarms software.

After a significant amount of simulation, the Peripheral Control software and the Carrier Group Alarms software were redesigned to perform their computations significantly faster by reducing the number of required database operations. A battery of tests seemed to indicate that the redesigned software prevents queue overflow and process abortion.

The following two sections describe our application of two automatic verification tools to these software designs.

REAL-TIME VERIFICATION

Model checking is a method of automatically verifying concurrent systems in which a finite-state model of a system is compared with a correctness requirement. The early model checkers used a temporal logic called Computation Tree Logic (CTL) for specifying correctness requirements, and employed enumerative algorithms for verification [7, 23]. The applicability of model checking paradigm was greatly enhanced due to the introduction of symbolic algorithms for verification [21]. Research in recent years has led to the development of model checkers that can analyze real-time systems [2, 3, 10, 15, 18].

Overview of COSPAN

We consider the tool COSPAN developed at Bell Laboratories. In the following, we briefly review the relevant aspects of COSPAN (see [17] for the underlying theory). We first describe the formalism for modeling state machines, and next consider the extension with real-time constraints.

Modeling Language

The system to be verified is modeled as a collection of coordinating processes described in the language S/R. Each process has two types of variables: *state* variables and *selection* (or output) variables. A state of a process P is an assignment of values of its state variables, and a selection of P is an assignment of values to its selection variables. A *global state* consists of states of all the processes and a *global selection* consists of selections of all the processes. The declaration of a process P contains the type declaration of all its variables, a set of possible initial states, and a transition relation specified as a set of update commands for all of the variables of process P. The update command for a selection variable specifies the set of possible values as a function of the current state of P and selections of some other processes. The update command for a state variable specifies the set of possible next states as a function of the current state of P and the current global selection. Thus, selection variables are used for communication among processes. The execution of an S/R model proceeds in rounds. In round 0, initial values of all the state variables are chosen. In each subsequent round, first the global selection is determined (which depends on the current global state), and then the next global state is determined (which depends on the current global state and the global selection). Thus, an execution of the model M produces an infinite sequence r of the form $s_0 \xrightarrow{\sigma_0} s_1 \xrightarrow{\sigma_1} s_2 \xrightarrow{\sigma_2} \cdots$ of global states s_i and global selections σ_i. *Acceptance conditions* are used to rule out uninteresting executions, for instance, to enforce fairness in resolving a nondeterministic choice. With the model M, we associate the set $L(M)$ consisting of all the sequences that satisfy the acceptance conditions.

Verification

Once the system under design is described as an S/R model M, it can be checked for different correctness conditions. The property to be checked is described as another process T whose acceptance conditions classify the executions of M in sets "good" and "bad," and all the good executions are removed from the language. Let N be the new model that contains the original model M together with the property T (technically, N is the product automaton $M \otimes T$). The model M satisfies the property T if $L(N)$ is empty. For instance, the user can check whether all states appearing in all executions of the model satisfy a state-predicate φ, that is, whether φ is an invariant of the model. The language S/R provides pre-defined macros for many of the commonly used properties such as invariants, absence of deadlocks, and liveness (e.g. whether a request is always followed by a response).

As just described, the verification problem is phrased as a language-emptiness question. Given an S/R model N, COSPAN checks whether $L(N)$ is empty using efficient algorithms. The algorithms supported include on-the-fly enumerative search and symbolic search using binary decision diagrams (BDDs). If the language is non-empty, then an execution is reported as evidence. For instance, if we are trying to verify whether φ is an invariant of the model, then COSPAN reports either an affirmation or an execution that leads to a state that violates φ.

Real-time Constraints

The S/R model is synchronous: the computation of all processes occurs in lock-step and is modeled by rounds. However, asynchrony can be modeled using delays and non-determinism. Intuitively, whenever an asynchronous process changes its state, it waits a nondeterministic number of rounds before proceeding. Every asynchronous process P has an additional state variable called *the pause-bit*. Whenever the state of P changes, its pause-bit is set to 1. In each subsequent round, the pause-bit may stay unchanged, or may be reset to 0. The selection of P, and hence, the next state, depends on the current value of the pause-bit. Pause-bits of different processes may stay set for different numbers of rounds, thereby introducing asynchrony.

In S/R, real-time constraints are expressed by associating lower and upper bounds with local states of asynchronous processes. These bounds limit the duration for which the pause-bit stays set in a given local state of a process. An execution is *timing-consistent* if all the global states can be assigned real-valued time-stamps that satisfy all the specified bounds. The semantics of a timed S/R model M with a table B of bounds is then the set $L(M, B)$ of its timing-consistent executions (see [3] for a formal definition). For instance, consider two asynchronous processes P and P', and suppose both simultaneously transition to their respective local states s and s'. Suppose the upper bound associated with s is less than the lower bound associated with s', then the pause-bit of P is reset *before* the pause-bit of P' is reset for the execution to be timing-consistent.

To check the correctness conditions of a timed model, an additional process T that removes good behaviors is introduced as before. The timing verification problem corresponds to checking emptiness of the language $L(N, B)$, where N is the product $M \otimes T$. Note that $L(N, B)$ can be empty while $L(N)$ is not. For instance, a model M may not satisfy an invariant φ, but with an appropriate choice of upper and lower bounds, all timing-consistent executions may satisfy the invariant φ. The property T itself can have real-time constraints, and thus, it is possible to check correctness conditions such as bounded-response (eg. whether a request is followed by a response within a specified period).

Real-time Verification

For real-time verification, COSPAN checks emptiness of the language $L(N, B)$ by constructing another automaton A_B, also as a S/R process. This process, when composed with the original model, rules out behaviors that do not satisfy the timing constraints: $L(N \otimes A_B)$ equals $L(N, B)$. The existence of such a finite-state constraining automaton A_B follows from the so-called region construction for timed automata [2]. The implementation in COSPAN supports a variety of heuristics to improve efficiency of this method. The states of the constraining automaton correspond to sets of values for timers, and may be either regions or convex unions of regions called zones. Once the definition of the constraining automaton has been generated, the original verification engine that tests emptiness of an S/R model can be used as a black-box, and the actual search can be done either enumeratively or symbolically using BDDs. These choices are coupled with an iterative solution which involves generating successive approximations to the constraining automaton. Furthermore, the underlying continuous semantics of time can be approximated in a conservative way by the integers, and this also provides a heuristic simplification of the timing analysis. An overview of the timing verification in COSPAN appears in [3].

Specification of the Design

The first step towards analysis of the software is the description of the design in S/R. We have constructed two models, one capturing the old design and one capturing the upgraded design.

Each model consists of several interacting processes: the PC process that models the peripheral control software, the INP process that models the input buffer, the SCHED process modeling the scheduler, and finally, the ENV process that issues the alarms. A brief description of these processes follows.

- The ENV process models the environment that issues the alarms. The alarms are issued periodically; the duration of the period is fixed. The number of alarms issued in each period varies, and is chosen nondeterministically from a given range.
- The process INP models the buffer used to store the alarms. It interacts with ENV and with PC: whenever ENV issues alarms, the number of pending alarms is increased, and whenever PC removes alarms, the number of pending alarms is decreased. The buffer has a fixed size. While this buffer is implemented as a queue in the switch software, for the chosen level of abstraction for us, the actual entries in the buffer are irrelevant. So we model the buffer by a single (bounded) integer variable that tracks the number of pending alarms.
- The main process, called PC, is the software that samples the alarms in INP and stores them in the output queue. The PC process has a boolean selection variable READY that indicates whether it needs to be scheduled. READY is true if there are any pending alarms in INP and false otherwise. Once READY is set and PC receives the appropriate signal from SCHED, it is activated. After initialization, it chooses a set of alarms to process from the buffer INP. The number of alarms chosen is determined by a (simple) algorithm that depends on the current number of pending alarms. The next step is the computation phase. It should be noted that we describe the design at a very high level of abstraction. For instance, all the computation required to sample the hardware alarms is abstracted, retaining only the time required for this computation. The time required for computation depends upon the number of alarms processed, and is different in the old and the new models. Once the computation is finished, INP is decremented according to the number of alarms that have been processed, and the PC process signals to SCHED to give up its time-slice. In the switch software the processed alarms are stored in an output queue, but we do not model the output queue explicitly.

- The scheduler process SCHED controls the invocation of the PC process, and models the real-time scheduling assumptions in the switch. We assume that the PC process has the highest priority; hence, every time the scheduler allocates the next time-slot, PC gets the priority if READY is true. Since the PC process has high priority, it is not considered a design violation for it to retain control past its recommended time-slice; however, if it exceeds its maximum allocated time-slice, it will be killed by the scheduler process. We assume all other processes in the switch are well-behaved, and thus, will relinquish control within their recommended time-slice. Observe that the modeling of the scheduler abstracts the interaction of the PC with the rest of the switch.

We check the models for two properties: buffer overflow and process abortion. The buffer overflow condition occurs when the input buffer to the PC process exceeds its size. The process abortion condition occurs when the time taken by the PC to finish its computation in one invocation exceeds the maximum allowed time-slice. Both these conditions can be specified as simple invariants in S/R.

Analysis

As explained earlier, COSPAN uses the language-emptiness test to detect executions of the model that violate the property. If such an execution is detected then it is reported as a counter-example. For the properties that we checked, the counter-example is a sequence of global states, starting with an initial state, leading to a state in which the buffer overflows or the process gets aborted.

As expected, the model capturing the old design does not satisfy either property, and the verifier reported a counter-example as an evidence. This is consistent with the problems discovered during testing. In this case, both buffer overflow and process abortion are possible.

For the upgraded design, while the process abortion condition does not occur, the verifier reported that the buffer *can* overflow, and produced a counter-example involving several thousand steps of the model. Thus, to our surprise, the new design also does not satisfy the correctness requirements. We are currently working with the software developers to reproduce these conditions in testing of the switch software in the lab environment, and to analyze possible solutions.

It is easy to change parameters of the model such as the size of the buffer, the maximum number of alarms per interval, and the duration of the interval between successive arrivals, and to test which combinations satisfy the properties and which do not. This can be useful feedback to the designers. For instance, for the new model, we found the maximum number of alarms that can be permitted without leading to the buffer overflow condition; this computation was performed by iteratively changing the model and repeating the analysis.

For this example, among the various options available in COSPAN, the use of regions for the constraining automaton and the symbolic search using binary decision diagrams is computationally least expensive. The computational requirements vary depending upon the choice of the parameters: the number of states explored ranges between 10^4 to 10^6, the amount of memory required ranges between 1MB to 20MB, and the CPU time required ranges between 5 to 5000 seconds on an SGI machine with 12 150MHz IP19 processors and 1280 MB of main memory.

PROBABILISTIC VERIFICATION

The models of the previous section assume that 5ESS processes relinquish control within their recommended time-slice. Under this assumption, functional and real-time properties can be proved of the PC software using verification tools such as COSPAN. In the context of a large system such as the 5ESS switch, this parallels the best form of reasoning feasible by individual developers: namely, that their software behaves correctly under the assumption that software developed by others behaves well.[1]

In reality, however, all other processes do not necessarily relinquish control within the recommended time-slice. For example, the PC process itself does not! (Note however, in the new design, it does relinquish control within its maximum allocated time-slice and hence is not aborted.) This behavior is intentional since the PC process has high priority; in the case of other processes, such behavior may be unintentional and hence undocumented. Thus, in order to prove properties about the reliability of the 5ESS software, it would also be desirable to take into account the *probability* that other software processes violate their recommended time-slices.

Probabilities arise in two other ways in our application. First, since our COSPAN analysis showed that both the old and new software designs can result in queue overflow, it would be desirable to show that the new software design is in fact *better than* the old design in the sense that it is *less likely* to result in queue overflow. Second, it would also be desirable to model the probability distribution of the incoming rate of alarms.

[1] Typically, individual software components in the 5ESS software exceed their real-time requirements only when the switch is experiencing overload conditions. The architecture of the switch software has been designed to ensure overall reliability of the switch even under such conditions: for example, some processes are throttled during peak calling hours.

Criteria for a Probabilistic Real-Time Model
In recent years, there has been growing research on integrating probabilistic and real-time verification, and numerous probabilistic real-time process algebras, automata, temporal logics, and algorithms have been developed (e.g. [1, 19, 9]). In particular, most of these approaches extend labeled transition systems and automata by allowing probability distributions to be placed on the transitions, corresponding to the probability that the given transition is selected from its parent state. The associated logics give the probabilities with which temporal properties are satisfied by the specifications. However, most of this work has been theoretical, and few tools have been developed to support these formalisms.

Since COSPAN does not support probabilistic reasoning, we were interested in an approach that would allow us to add probabilities to our real-time models. The following criteria are important for such an approach in the context of our application:

- Verification should support temporal logic extended with real-time and probabilistic information, and *the theory should be supported by a toolset* so that practical applications can be verified. This has implications for our choice of the real-time model. In particular, we are not aware of the existence of any tools that support continuous-time and probabilities, and that satisfy all the following criteria. Hence, we decided to limit ourselves to models in which time is assumed to pass discretely.
- There must be a way to compositionally specify concurrent processes, so that the model reflects the architecture of the software application. For example, it must be possible to specify the PC process and the scheduler process independently.
- There must be a way to model the passage of time independently from the rest of the specification, for example using clocks. In particular, time should *not* be modeled as one time unit elapsing per transition, as different events typically take different amounts of time. For example, the time needed to retrieve data from a database may vary depending on the structure of the data.
- It must not be necessary to associate probabilities with individual events, since it is difficult to estimate the probabilities with which individual events occur.
- It must be possible to assign probabilities to timing delays, so that we can specify, for example, the probability that a process exceeds its recommended time-slice by n milliseconds.
- It must be possible to specify geometric probability distributions, since the inter-arrival time between alarms is conveniently modeled as a geometric distribution (assuming independence of alarms).

TPWB: Timing and Probabilities Workbench
We discovered that the Timing and Probabilities Workbench (TPWB) [11, 13] satisfies all of our above criteria. The modeling language in TPWB is TPCCS – a version of the Calculus for Communicating Systems (CCS) [22], and the logic in TPWB is TPCTL – a version of Computation Tree Logic (CTL) [7]. Both the modeling language and logic extend their respective theories with real-time and probabilistic information.

Like CCS, TPCCS is a process-algebra describing labeled transition systems. Unlike standard labeled transition systems, however, labeled transition systems in TPCCS have two kinds of states: reactive states and probabilistic states. Transitions from reactive states are labeled only with events. Transitions from probabilistic states are labeled only with probabilities; these specify the probability that the given transition is traversed from its parent state. Finally, the passage of (discrete) time is modeled as a distinguished event in TPCCS; hence time-labeled transitions can only emanate from reactive states.

Like CTL, TPCTL is a branching time logic for describing the behavior of some or all paths in a labeled transition system specification. For example, the property that "in all paths of the system, the queue does not overflow" can be described as a CTL property. In addition, TPCTL allows the quantification of such properties over real-time: for example, the property that "in all paths of the system, the queue does not overflow within 20 milliseconds from the initial state of the system" can be described in TPCTL. Finally, TPCTL extends properties with probabilistic information. For example, in TPCTL one can state the property that "in all paths of the system, with 95 percent probability, the queue does not overflow within 20 milliseconds from the initial state of the system." Thus, properties regarding the reliability of systems can be formulated in TPCTL.

The TPWB supports automatic verification of TPCTL formulas on specifications described in TPCCS. It also supports bisimulation equivalence checking of TPCCS specifications, but we have not used this facility for our application.

Specification of the Design
We first wrote two models in TPCCS: one describing the old 5ESS design and the other describing the upgraded design. These models are very similar to their S/R counterparts, the main extension being that probabilistic information is added to the incoming alarm rate

and the scheduler. The other main difference is that computational requirements of the TPWB forced us to significantly scale down many of the parameters, such as the number of incoming alarms, the size of the INP buffer, and the computation time of the PC process.

The TPCCS models consist of the following interacting processes; we describe below the differences between these processes and those in our S/R models.

- The ENV process models the environment that issues the alarms. The important difference from its S/R counterpart is that the number of alarms issued in each period varies within a given range according to a uniform probability distribution.
- The process INP models the buffer. It is essentially the same as its S/R counterpart, the only differences being in some minor implementation details. For example, since the version of TPCCS we used does not support value-passing, we modeled the buffer as a concurrent TPCCS process with increment/decrement/overflow events, rather than as a single integer variable as in S/R.
- The main process, called PC, is the software that samples the alarms in INP and stores them in the output queue. Again, it is essentially the same as its S/R counterpart, and the time required for its computation is different in the old and the new models.
- The scheduler process SCHED controls the invocation of the PC process, and models the real-time assumptions in the switch. The important difference from its S/R counterpart is that other processes are not necessarily assumed to relinquish control within their recommended time-slice, but only within their maximum allocated time-slice (after which they will get killed by the scheduler anyway). Furthermore, the time taken by other processes is specified by a uniform probability distribution.

We note that in our TPCCS models, one time instant represents one millisecond of elapsed time.

We first checked some sanity conditions of our models to ensure that their functional and timing behavior corresponds to some of our expectations:

- With probability 1, the PC process is allowed to run for its maximum allocated time-slice without getting killed by the scheduler.
- With probability 1, the PC process gets killed by the scheduler if it does not relinquish control within its maximum allocated time-slice.

Time (millisec)	Old Model (probability)	New Model (probability)
20	1	1
50	1	1
59	1	1
60	.9998	1
70	.9992	1
80	.9985	1
90	.9957	1
100	.9943	1
150	.9796	1
200	.9631	1

Table 1: Avoidance of PC Process Abortion

- With probability 1, the PC process is only scheduled if it is ready (i.e. if alarms exist).
- With probability 1, if an alarm arrives then the PC is eventually scheduled, or is currently running and either voluntarily relinquishes control or is killed.
- An alarm will eventually arrive. (This is due to the alarm inter-arrival time being modeled as a geometric distribution.)

As with our COSPAN application, we then checked both models for the two main properties: buffer overflow and process abortion. In contrast to COSPAN, however, we also computed the likelihood that each of two models will satisfy the properties within particular time bounds.

Analysis

As mentioned earlier, we had to significantly scale many of the parameters of the models in order to effectively use the TPWB. This scaling resulted in non-integer values for some parameters, which consequently had to be rounded off. This compromises the accuracy of the model with respect to the actual design. However, we have still obtained some useful feedback from our analysis.

After scaling, our model of the old 5ESS software design has approximately 8000 states, while the model of the new design has approximately 6000 states. We used TPWB to show that both models are deadlock-free and have no "non-Zeno" behavior; that is, both models only permit finite computations within bounded amounts of time. Furthermore, the TPWB shows that both models satisfy all of the sanity conditions above.

We also used the TPWB to show that the old model can result in process abortion and queue overflow, as expected. Furthermore, as with its S/R counterpart, the new model is guaranteed to prevent process abortion, but still may result in queue overflow. In addition to checking the possibility of these violations, we also

Time (millisec)	Old Model (probability)	New Model (probability)
20	1	1
30	1	1
39	1	1
40	.9999	.99997
50	.9992	.99995
60	.9982	.99994
70	.9970	.99991
80	.9955	.99991
90	.9937	.99988
100	.9919	.99986
150	.9808	.99981
200	.9683	.99970

Table 2: Avoidance of Queue Overflow

used the TPWB to compute the likelihood of these conditions being satisfied by the models within certain time bounds. Our results are given in Table 1, which lists the probabilities for the avoidance of process abortion, and Table 2, which lists the probabilities for the avoidance of queue overflow. The information in the tables should be interpreted as follows. Suppose for a given model and given time t, p is the probability listed in a table. Then, on all paths of the model, the property is satisfied for t milliseconds with probability p. For example, Table 1 specifies that, on all paths of the old model, with probability .9998 the PC process will not be aborted within 60 milliseconds. Since the new model is guaranteed not to have process abortion, it satisfies the property with probability 1 on all paths for all finite lengths of time.

The tables show that, for at least up to 200 milliseconds, the new model satisfies both the process abortion avoidance and queue overflow avoidance properties with greater probability than the old model. Thus, in a precise formal sense, the new design can be regarded as an improvement over the old one.

Ideally, we would have continued this analysis for much longer real-time segments. However, even for the 200 millisecond analysis, we required upwards of 24 hours of CPU time and 200 MB memory on a lightly loaded SGI Challenge XL with 4 MIPS R4400 150MHz processors and 512 MB of main memory size. Thus, more detailed analyses seem infeasible at the current time.

Similar limitations of the TPWB tool prevent us from reducing the significant scaling we did on our TPCCS models. We note that the modeling of probability distributions over timers also seems to significantly affect the state space of the model, and hence the performance of the TPWB tool. We had originally specified uniform probability distributions over timers as labeled transition systems with no branching after the root node. For example, a timer that can fire between 0 and 3 time instants with uniform probability was modeled as a labeled transition system whose root contains four outgoing transitions, each labeled with probability 1/4. The first transition is succeeded by a path (with no subsequent branching) modeling a timer that fires immediately, the second transition is succeeded by a path (with no subsequent branching) that models a timer that fires after 1 time instant, and so forth. Thus, the choice of the timer duration is made at the root.

We later changed our model of probability distributions over timers to be binary-branching labeled transition systems, where the choice of timer duration is made after every time instant, rather than at the root node. The above example is then modeled as a labeled transition system with two outgoing transitions, one labeled with probability 1/4 and the other with probability 3/4. The first transition is succeeded by a timer that fires immediately. The second transition is succeeded by time-passing transition; this in turn leads to a binary-branching node whose first transition is labeled with probability 1/3 and second transition is labeled with probability 2/3. The first transition again is succeeded by a timer that fires immediately; the second transition again by a time-passing transition and then by a binary-branching node, and so forth. Hence, the timer duration can be between 0 and 3 time instants with uniform probability; however, the choice of remaining duration is made after every time instant.

Our experiments indicated that the binary-branching technique significantly reduced the size (of the state spaces) of TPCCS processes, and hence improved the performance of the TPWB tool. We believe that this optimization is correct for our purposes for the following reasons. First, we conjecture that the modified model has the same linear traces as our original model (i.e. the models are language-equivalent). Second, all of the properties we verified seem to be contained in a linear-time subset of TPCTL. In the usual untimed setting, linear-time properties – even when expressed in a branching-time logic – respect language equivalence. We conjecture that this result generalizes to TPCTL/TPCCS as well, and hence that both models satisfy the same subset of our properties.

Performance Analysis

The TPWB also provides an interface to the Generalized Timed Petri Net Analyzer (GTPNA) [16], which supports performance analysis on a stochastic extension of Petri Nets. Some interesting properties that can be proved using GTPNA include average throughput and resource utilization, both of which are interesting in our application.

The TPWB transforms a restricted subset of TPCCS into equivalent Generalized Timed Petri Nets. Using the TPWB and GTPNA together, one can thus prove both worst-case temporal properties and average-case performance properties on a single model written in TPCCS. Unfortunately, due to the non-determinism in the incoming alarm rate, our TPCCS models do not fall into the class currently supported by the TPWB tool, and hence we could not use the GTPNA tool in conjunction with our TPCCS models.

BENEFITS

Based on our experience in using COSPAN and TPWB in the context of this 5ESS application, we believe that these tools have many potential benefits in software development. From our experience in using other automatic verification techniques, we are convinced that these benefits also hold for most current verification techniques and tools.

- *Automatic verification can be used to find subtle errors in software designs.*

  Automatic verification performs an exhaustive state-space exploration and considers *all* possible interactions of the concurrent processes in large real-time systems; these interactions often lead to subtle bugs that are extremely difficult to discover using the conventional techniques of simulation and testing.

  We note that verification of software designs does *not* necessarily prove the *absence* of bugs in the target software, since the design may abstract away important details. This is not necessarily a limitation of automatic verification, since "finding the last bug in software" is a well-known myth [6].

  Automatic verification is, however, a very powerful tool in the *detection* of bugs, and can be regarded as a sophisticated form of testing. Furthermore, it provides a form of regression testing: verification tools can automatically check that new versions of the software designs continue to satisfy the desired properties that held of the original design.

- *Errors can be caught earlier in the software development cycle.*

  In contrast to the conventional techniques of simulation and testing, automatic verification finds bugs in software designs as opposed to actual implementations. Hence, bugs can be found at design-time, an early stage of software development, rather than at the late development stage of testing. As is well-known, the cost of correcting software errors is a function of the phase in which corrections are made [5]. In particular, it is far cheaper to fix a design error at design-time rather than after implementation has been completed and testing has begun. Fixing the error at this late stage may entail significant re-implementation and re-testing of the software. For example, significant effort was needed to correct the actual 5ESS application in the context of the upgraded carrier group hardware.

- *Formalisms used in automatic verification support the modeling of designs at a high-level of abstraction.*

  Formal specification languages can be used to describe software designs at a high-level of abstraction, allowing the omission of low-level information. This can result in designs that are easier to understand and maintain. For example, in our models, we abstracted all the computation required to sample the hardware alarms and retained only the time required for this computation. This was sufficient for our purposes since only the timing requirements, and not the details of the computation, led to the real-time difficulty in the software.

- *Automatic verification can be used to effectively generate test cases.*

  Upon discovery that the software design violates a desired property, typical automatic verification tools based on model-checking provide scenarios as evidence of the violation. By mapping from the abstract events in the model to the actual events in the implementation, these scenarios can be used as test-cases on the actual software implementation. Since these scenarios can often be quite subtle – comprised of several hundreds or thousands of state-changes – the corresponding test-cases are extremely difficult to devise by hand. For example, the scenario generated by COSPAN consists of several thousand states.

- *Automatic verification can aid in the maintenance of real-time software.*

  Empirical observations have shown that a significant percentage of errors can occur in redesigns during software maintenance [12]. Upgrades to hardware or processors during the maintenance of real-time systems may introduce real-time difficulties, the correction of which often require the execution of some concurrent processes to be speeded up. Two problems immediately arise, as with our 5ESS application. First, the original software has typically been engineered to be quite efficient in the first place. Thus, even speeding it up by a very small amount can be extremely difficult and may entail significant restructuring of the software architecture. Second, the amount of speedup required to resolve the real-time difficulty is not known prior to testing. Therefore, a coarse estimation of the required speedup needs to be made while modifying the software. Testing may reveal that the speedup was not sufficient, requiring further modifications of the software and further iterations of testing.

In contrast, formal specifications can be parameterized with respect to the real-time behavior of processes, and automatic verification can be easily repeated on models with different instantiations of these parameters. For example, in our COSPAN models, we changed parameters such as the size of the buffer, the maximum number of alarms per interval, and the duration of the interval between successive arrivals, and tested which combinations satisfy the desired properties and which do not.

- *Automatic verification can support reasoning about the reliability of real-time systems.*

  Using the combination of probabilistic and real-time verification, properties can be proved about the reliability of real-time systems. For example, we were able to show in our TPCCS models that the new software design satisfies the desired properties with higher probability than the older software design, and hence is more reliable.

- *The verification of temporal properties and average-case performance analysis can be combined within a single formalism and tool-set.*

  Typically, automatic verification techniques together with temporal logic support reasoning about worst-case behavior of programs. For example, we proved properties regarding the possibility of process abortion and queue overflow in our S/R models, and the possibility and likelihood of these events in our TPCCS models.

  In addition, average-case properties generally supported by performance analysis tools are also interesting in the context of real-time systems. For example, it would be useful to calculate the average throughput and resource utilization for our application.

  At present, specifications must typically be described in different formalisms in order to use both automatic verification tools and performance analysis tools. The drawback is that there is generally no formal connection between these formalisms, and hence no guarantee that the specifications are equivalent in any rigorous sense. Recently, there has been some promising work on combining these approaches [11, 13, 4].

LIMITATIONS

During the course of our application, we have observed several limitations of the COSPAN and TPWB tools. From our experience in using other automatic verification techniques, we strongly believe that these limitations hold for most current software[2] verification techniques and tools.

- *Expertise with the specification formalisms and the verification algorithms is needed in order to use the tools effectively.*

  We found that formulating suitable abstractions of this application in S/R and TPCCS was subtle. If the synchronization operations and modeling of time passage in these formalisms are not used with care, the models can have subtle errors. Furthermore, knowledge of the model-checking algorithms can greatly affect the performance of the tools. For example, the use of regions for the constraining automaton and the symbolic search using binary decision diagrams is computationally least expensive for this COSPAN application. Concurrent timers must also be used with care to avoid state-space explosion in COSPAN. Similarly, the modeling of uniform distributions using binary branching trees significantly reduced the state space of our TPCCS models.

- *The specification may not be faithful to the software it is intended to model.*

  By definition, the specification formalisms are intended to abstract low-level information from the target software. Even if the specification formalism is used correctly, the abstractions themselves may or may not be correct. Thus, the verification of a specification does not necessarily guarantee that the target software is correct.

  Similarly, the specified probability distributions may not reflect the actual system or environment. For example, we have assumed a uniform probability distribution for both the incoming alarm rate and the process scheduling delays, and a geometric probability distribution for the alarm inter-arrival rate. This is our best guess of the behavior of the actual carrier group alarm hardware and the 5ESS scheduler process. Since our analysis is highly dependent on these distributions, even slightly incorrect modeling of the actual distribution renders the analysis incorrect. This is a limitation of performance analysis tools in general.

- *Verification techniques are computationally expensive and hence use a great deal of space and time.*

  The state-space explosion problem is well-known in the verification community. This problem is even

[2] Some of these limitations are in contrast to hardware verification, where the verification formalisms more closely correspond to the actual hardware.

more acute in real-time verification due to the use of concurrent timers. COSPAN performed efficiently on our particular model; however, in the future, upgraded hardware may again significantly increase the possible rate of incoming 5ESS alarms. Our expectation – based on our other experiences using COSPAN – is that such significant upward scaling would result in state-space explosion problems.

For our TPWB model, we had to significantly scale our design; even with this scaling, we could not prove many potentially interesting timing properties of our models due to space and time limitations of the tool. More research is needed to develop efficient algorithms and heuristics for probabilistic verification.

ACKNOWLEDGMENTS

We are grateful to Ed Knappe for his help in performing tests in the 5ESS lab environment. We thank Mark Ardis and Glenn Bruns for comments on this paper, and Bob Kurshan and Sandy Wiedemeier for discussions about this work. We thank Lars-åke Fredlund for discussions and help with TPWB.

REFERENCES

[1] R. Alur, C. Courcoubetis, and D. Dill. Model-checking for probabilistic real-time systems. In *International Colloquium on Automata, Languages and Programming*, pages 115–126, 1991.

[2] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[3] R. Alur and R. Kurshan. Timing analysis in COSPAN. In *Hybrid Systems III*, LNCS 1066. Springer-Verlag, 1996.

[4] M. Bernardo, L. Donatiello, and R. Gorrieri. Integrating performance and functional analysis of concurrent systems with EMPA. Technical report, Dept. of Comp.Sci., Univ. of Bologna, 1996.

[5] B. Boehm. *Software Engineering Economics.* Prentice Hall, 1981.

[6] F. Brooks. *The Mythical Man-Month: Essays in Software Engineering.* Addison-Wesley, 1995.

[7] E. Clarke and E. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proc. Workshop on Logic of Programs, LNCS Volume 131*, pages 52–71, 1981.

[8] E. Clarke and R. Kurshan. Computer-Aided Verification. *IEEE Spectrum* **33** *(6)*, pages 61–67, (1996).

[9] C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. In *Symp. on Foundations of Comp. Sci.*, pages 338–345, 1988.

[10] C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool kronos. In *Hybrid Systems III: Verification and Control*, LNCS 1066, pages 208–219, 1996.

[11] L. Fredlund. The Timing and Probability Workbench: A tool for analysing timed processes. Technical report, Department of Computer Systems, Uppsala University, 1994.

[12] R. Grady and D. Caswell. *Software Metrics: Establishing a Company-Wide Program.* Prentice Hall, 1987.

[13] H. Hansson. *Time and Probability in Formal Design of Distributed Systems.* Series in Real-Time Safety Critical Systems. Elsevier, 1994.

[14] G. Haugk, F. Lax, R. Royer, and J. Williams. The 5ESS(TM) switching system: Maintenance capabilities. *AT&T Technical Journal*, 64(6 part 2):1385–1416, July-August 1985.

[15] T. Henzinger, P. Ho, and H. Wong-Toi. HyTech: the next generation. In *TACAS 95: Tools and Algorithms for the Construction and Analysis of Systems*, LNCS 1019, pages 41–71, 1995.

[16] M. Holliday and M. Vernon. The GTPN analyzer: numerical methods and user interface. Technical report, Department of Computer Science, University of Wisconsin – Madison, 1986.

[17] R. Kurshan. *Computer-aided Verification of Coordinating Processes: the automata-theoretic approach.* Princeton University Press, 1994.

[18] K. Larsen, P. Pettersson, and W. Yi. Compositional and symbolic model-checking of real-time systems. In *Proceedings of the 16th IEEE Real-Time Systems Symposium*, 1995.

[19] G. Lowe. *Probabilities and Priorities in Timed CSP.* PhD thesis, Oxford University, 1993.

[20] K. Martersteck and A. Spencer. Introduction to the 5ESS(TM) switching system. *AT&T Technical Journal*, 64(6 part 2):1305–1314, July-August 1985.

[21] K. McMillan. *Symbolic model checking: an approach to the state explosion problem.* Kluwer Academic Publishers, 1993.

[22] R. Milner. *Communication and Concurrency.* Series in Computer Science. Prentice Hall, 1989.

[23] J. Queille and J. Sifakis. Specification and verification of concurrent programs in CESAR. In *Proceedings of the 5th International Symposium on Programming*, LNCS 137, pages 195–220, 1982.