Computing Quantitative Characteristics of Finite-State Real-Time Systems *

S. Campos, E. Clarke, W. Marrero, M. Minea
School of Computer Science
Carnegie Mellon University

H. Hiraishi
Dept. of Information and Communication Sciences
Kyoto Sangyo University

Abstract

This paper presents a general method for computing quantitative *information about finite-state real-time systems. We have developed algorithms that compute exact bounds on the delay between two specified events and on the number of occurrences of an event in a given interval. This technique allows us to determine performance measures such as schedulability, response time, and system load. Our algorithms produce more detailed information than traditional methods. This information leads to a better understanding of system behavior, in addition to determining its correctness. The algorithms presented in this paper are efficiently implemented using binary decision diagrams and have been incorporated into the SMV symbolic model checker. Using this method, we have verified a model of an aircraft control system with 10^{15} states. The results obtained demonstrate that our method can be successfully applied in the verification of real-time system designs.*

1 Introduction

A number of algorithms have recently been proposed for verifying the behavior of finite-state real-time systems [1, 4, 6, 7, 8]. These algorithms assume that timing constraints are given explicitly in some notation like temporal logic. Typically, the designer provides a constraint on response time for some operation, and the verifier automatically determines if it is satisfied or not. Unfortunately, these techniques do not provide any information about how much a system deviates from its expected performance, although this information can be extremely useful in fine-tuning the behavior of the system.

*This research was sponsored in part by the National Science Foundation under grant no. CCR-9217549, by the Semiconductor Research Corporation under contract 92-DJ-294, and by the Wright Laboratory, Aeronautical Systems Center Air Force Materiel Command, USAF, and the Advanced Research Projects Agency (ARPA) under grant number F33615-93-1-1330. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied of the U.S. government.

In this paper we give algorithms to compute quantitative timing information, such as exact upper and lower bounds on the time between a request and the corresponding response. Our algorithms provide insight into *how well* a system works, rather than just determining whether it works at all. They enable a designer to determine the timing characteristics of a complex system given the timing parameters of its components. This information is especially useful in the early phases of system design, when it can be used to establish how changes in a parameter affect the global behavior of the system.

We model a real-time system as a labeled state-transition graph, where each path corresponds to an execution trace of the actual system. This graph is implemented internally using binary decision diagrams(BDDs) [2], which generally produce a very compact representation. By employing symbolic model-checking techniques [3, 12], we are able to handle extremely large state spaces with up to 10^{30} states efficiently. We show how to determine the minimum and maximum length of all paths leading from a set of starting states (representing the request) to a set of final states (representing the response). We also present algorithms that calculate the minimum and the maximum number of times a specified condition can hold on a path from a set of starting states to a set of final states. These algorithms are also extended to *timed transition graphs* (TTG) [4], a model in which transitions take more than one time unit to occur. We believe that the techniques developed can be adapted to other models of computation as well.

Other approaches for analyzing real-time system exist. For example, the *rate monotonic scheduling* theory (RMS) [9, 10, 13] defines a priority assignment algorithm that guarantees optimal response time [10]. The RMS theory proposes a schedulability test based on total CPU utilization; a set of processes (which have priorities assigned according to RMS) is schedulable if the total utilization is below a computed threshold. If the utilization is above this threshold, schedulability is not guaranteed. Moreover, this analysis only considers certain types of processes with limitations, for example, on periodicity and synchronization. Another approach to schedulability analysis uses al-

gorithms for computing the set of reachable states of a finite-state system [7, 8]. The algorithms construct the model with the added constraint that whenever an exception occurs (e.g. a deadline is missed) the system transitions to a special exception state. Verification consists of computing the set of reachable states and checking whether the exception state is in this set. No restrictions are imposed on the model in this approach, but the algorithm only checks if exceptions can occur or not.

We develop an analysis method that does not impose any restriction except that the system be modeled as a set of processes that run in parallel and are defined by state-transition graphs. For example, the actual functional behavior of each process can be modeled and analyzed. Schedulability is determined by computing the minimum and maximum execution times for all processes. The process set is schedulable if and only if each process is guaranteed to finish execution before its next period starts. Our technique always determines if the set of processes is schedulable or not, unlike RMS analysis, which may not provide any schedulability information if utilization is above the computed threshold. If the processes are not schedulable, our algorithms determine which specific deadlines are missed and by how much. When no deadline is missed, the same results provide response times for each process, an important performance measure for real-time systems.

To demonstrate how our tools work, we verify a simplified aircraft control system. We model the software that controls the various components of an airplane, and gather timing information about the system using the tools described above. The system consists of a set of priority driven processes, where each process is responsible for a subsystem of the aircraft, such as navigation, display, radar and weapons. The schedulability of this process set is determined. Moreover the computation of quantitative characteristics also provides other valuable results about the system being modeled, such as:

- The overhead associated with preemption by other processes. This information is extremely important for determining the amount of priority inversion in a system.
- How fast a subsystem responds to an event. For example, in this model, pressing the fire button generates a complex sequence of events before the weapons are actually fired. We were able to determine the overhead imposed by the firing protocol and how it affects the overall response time of the system.

The different types of properties described above show how versatile this approach is. Many other quantitative characteristics can be computed by our algorithms. Moreover, in each case we were able to provide the user with insight into the behavior of the system, as opposed to only asserting its correctness. This information leads to a better understanding of system behavior and can be essential in improving performance.

2 Modeling Real Time Systems

The real-time systems we verify are modeled using state-transition graphs. A state $\overline{v}$ in this model can be thought of as a vector assigning values to the state variables $v_1, v_2, \ldots, v_n$. The transition relation $N(\overline{v}, \overline{v'})$ can be represented as a Boolean formula which evaluates to true when there is a transition in the model from the state $\overline{v}$ to the state $\overline{v'}$, where $\overline{v} = \langle v_1, \ldots, v_n \rangle$ and $\overline{v'} = \langle v'_1, \ldots, v'_n \rangle$. A *path* in the transition graph is defined as a sequence of states $\overline{v_0}, \overline{v_1}, \overline{v_2}, \ldots$ such that $N(\overline{v_i}, \overline{v_{i+1}})$ is true for every $i \geq 0$. In addition, we define a set of initial states, and all computations are performed on states reachable from this set.

A set of states can also be represented by a Boolean formula which evaluates to true if and only if its variables are assigned the values of the variables in a state in the set. Note that if $S(\overline{v})$ is a formula representing a set of states and $N(\overline{v}, \overline{v'})$ is a formula for the transition relation, the formula $\exists \overline{v}[S(\overline{v}) \wedge N(\overline{v}, \overline{v'})]$ represents the set of successors to states in $S(\overline{v})$. This operation can be thought of as a function mapping a set of states $S(\overline{v})$ to the set of its successors $S'(\overline{v'})$.

We use binary decision diagrams (BDDs) [2] to efficiently represent Boolean formulas and to manipulate them using the standard Boolean operations. Because of the close relationship between a Boolean formula, its BDD, and the set of states satisfying the formula, we identify these three entities. In particular, sets and set operations are more intuitive than boolean operations on formulas or BDD operations so we present our algorithms using sets, but the implementation uses BDDs and the corresponding BDD operations.

3 Quantitative Timing Algorithms

We first present the lower bound algorithm (figure 1). The algorithm takes two sets of states as input, *start* and *final*. It returns the length of (i.e. number of edges in) a shortest path from a state in *start* to a state in *final*. If no such path exists, the algorithm returns infinity. The function $T(S)$ gives the set of states that are successors of some state in S. The function T, the sets of states R and R', and the operations of intersection and union can all be easily implemented using BDDs.

The first algorithm is relatively straightforward. Intuitively, the loop in the algorithm computes the set of states that are reachable from *start*. If at any point, we encounter a state satisfying *final*, we return the number of steps taken to reach the state.

Next, we consider the upper bound algorithm (figure 2). This algorithm also takes *start* and *final* as input. It returns the length of a longest path from a state in *start* to a state in *final*. If there exists an infinite path beginning in a state in *start* that never reaches a state in *final*, the algorithm returns infinity. The function $T^{-1}(S')$ gives the set of states that

proc *lower* (*start, final*)
$i = 0$;
R =*start*;
$R' = T(R) \cup R$;
while $(R' \neq R \wedge R \cap final = \emptyset)$ **do**
$i = i + 1$;
$R = R'$;
$R' = T(R') \cup R'$;
if $(R \cap final \neq \emptyset)$
then return i;
else return ∞;

Figure 1: Lower Bound Algorithm

are predecessors of some state in S'. We also denote by *not_final* the set of all states that are not in *final*. As before, the algorithm is implemented using BDDs.

proc *upper* (*start, final*)
$i = 0$;
R =*TRUE*;
R' =*not_final*;
while $(R' \neq R \wedge R' \cap start \neq \emptyset)$ **do**
$i = i + 1$;
$R = R'$;
$R' = T^{-1}(R') \cap not\_final$;
if $(R = R')$
then return ∞;
else return i;

Figure 2: Upper Bound Algorithm

The upper bound algorithm is more subtle than the previous algorithm. A backward search from the states in *not_final* is more convenient in this case than a forward search. Proofs of both algorithms can be found in [5].

We have also developed algorithms that calculate the minimum and the maximum number of times a specified condition *cond* can hold on a path from a set of starting states to a set of final states. For this purpose, we define a new state-transition system, in which the states are pairs consisting of a state in the original system and a positive integer, denoting the number of states in *cond* that have been traversed on such a path. Thus, if the original state-transition graph has state set S, then the augmented state set will be $S_a = S \times \mathbf{N}$. The augmented transition relation $N_a \subseteq S_a \times S_a$ is defined in terms of the original transition relation $N \subseteq S \times S$ by incrementing the integer component k whenever a state in *cond* is traversed.

$$N_a(\langle s, k\rangle, \langle s', k'\rangle) =$$
$$N(s, s') \wedge (s' \in cond \wedge k' = k + 1 \vee s' \notin cond \wedge k' = k)$$

The algorithms use the augmented transition relation and the value of the counter component k to produce the desired information. We have applied the same technique to a more powerful model of real-time systems, timed transition graphs [4], in which the time taken by a transition is defined by a time interval. These extensions can also be found in [5].

4 Example — An Aircraft Control System

One of the most critical applications of real-time systems is in aircraft control. It is extremely important that time bounds are not violated in such systems. Because of the risks involved in the failure of an aircraft, only conservative approaches to design and implementation are routinely used. Many modern techniques for software design such as formal methods are not commonly employed. We believe that formal verification can be very useful in increasing the reliability of these systems by assisting in the validation of schedulability and response times of the various components.

This section briefly describes an aircraft control system used in military airplanes. Such a control system can be characterized by a set of sensors and actuators connected to a central processor. This processor executes the software to analyze sensor data and control the actuators. Our model describes this control program and determines whether its timing constraints are met. The requirements used are similar to those of existing military aircraft, and the model is derived from the one described in [11].

The aircraft controller is divided into systems and subsystems, each of which performs a specific task in controlling the airplane:

- Navigation: Computes aircraft position.
- Radar Control: Receives and processes data from radars. It also identifies targets and target position.
- Radar Warning Receiver: This system identifies possible threats to the aircraft.
- Weapon Control: Aims and activates aircraft weapons.
- Display: Updates information on the pilot's screen.
- Tracking: Updates target position. Data from this system is used to aim the weapons.
- Data Bus: Provides communication between processor and external devices.

Timing constraints for each subsystem are derived from factors such as required accuracy, human response characteristics and hardware requirements. The following table presents the subsystems being modelled, as well as their major timing requirements. In order to enforce the different timing constraints of the processes, priority scheduling is used. The priority assignment has been done according to the RMS theory [9, 10].

Concurrent processes are used to implement each subsystem. With the exception of the weapon system, all other systems contain only periodic processes. The weapon system contains a mixture of periodic and aperiodic processes. It is activated when the display keyset subsystem identifies that the pilot has pressed the firing button. This event causes the weapon protocol subsystem to be activated. It then signals the weapon aim subsystem that has been previously

System	Subsystem	Per.	Exec	%cpu	Pri
Display	status update	200	3	1.50	12
	keyset	200	1	0.50	16
	hook update	80	2	2.50	36
	graph. displ.	80	9	11.25	40
	store update	200	1	0.50	20
RWR	contact mgmt	25	5	20.00	72
Radar	target update	50	5	10.00	60
	track filter	25	2	8.00	84
NAV	nav update	50	8	16.00	56
	steer cmds.	200	3	1.50	24
Track	target update	100	5	5.00	32
Weapon	weapon prot.	200*	1	0.50	28
	weapon aim	50	3	6.00	64
	weapon rel.	200**	3	1.50	98
Dat Bus	poll device	40	1	2.50	68

* Weapon protocol is an aperiodic process with a deadline of 200ms.

** Weapon release has a period of 200ms, but its deadline is 5ms.

Subsystem	dead line	Execution Times			
		preempt		no preempt	
		min	max	min	max
Weapon release	5	3	3	3	9
Radar track filter	25	2	5	2	10
Contact mgmt.	25	7	10	7	15
Data bus poll	40	1	11	1	14
Weapon aim	50	10	14	2	18
Radar target upd	50	12	19	12	19
NAV update	50	20	34	20	27
Display graphic	80	10	44	10	43
Display hook upd	80	14	46	14	47
Track target upd	100	26	51	26	51
Weapon protocol	200	1	21	3	46
NAV steer cmds.	200	35	85	36	74
Display store upd	200	36	95	37	97
Display keyset	200	37	96	38	98
Display status upd	200	40	99	41	101

blocked. Weapon aim is then scheduled to be executed every 50ms. It aims the aircraft weapons based on the current position of the target. It also decides when to fire and then starts the weapon release subsystem. The firing sequence can be aborted until weapon release is scheduled, but not after this point. Weapon release then executes periodically and fires the weapons 5 times, once per second.

5 Verification of the Aircraft Control System

We have implemented this control system in the SMV language [12]. The SMV model checker has been used to verify its functional correctness, while its timing correctness has been checked using the quantitative algorithms described in this paper. In order to optimize response time, we have implemented a preemptive scheduler. However, preemptability is a feature that may not always be available. Non-preemptive schedulers are easier to implement, and allow for simpler programs but usually increase response time for higher priority processes. To assess the effect of preemption in our system we have also implemented a non-preemptive scheduler.

Using the model described above, we were able to compute the schedulability of the system. This is one of the most important properties of a real-time system. It states that no process will miss its deadline. In this example the deadlines are the same as the periods (except for the weapon release subsystem). We determine schedulability by computing the minimum and maximum execution times for each process and checking if they always finish before their deadline. The RMS theory checks for schedulability by computing the CPU utilization of the process set. It may not provide any schedulability information if the utilization exceeds a certain threshold. Our method however, is always able to determine schedulability. Moreover, it only requires that processes be modelled as state graphs, while RMS imposes restrictions on their behavior.

The following table summarizes the execution times computed by our algorithms for both the preemptive and non-preemptive schedulers. Processes are shown in decreasing order of priority. We can see from this table that the process set is schedulable using preemptive scheduling. An analysis of a similar process set using RMS showed that only the first eight processes were guaranteed to meet their deadlines [11]. From our results we can also identify many important parameters of the system. For example, the response time is usually very low for best-case computations, but it is also good for the worst case. Most processes take less than half their required time to execute. This indicates that the system is still not close to saturation, although the total CPU utilization is high.

Notice also that preemption does not have a big impact on response times. Except for the most critical process, all others maintain their schedulability if a non-preemptive scheduler is used. Moreover, we can see that non-preemption causes weapon release to miss its deadline, but by a relatively small amount. If a preemptive scheduler were expensive, reducing the CPU utilization slightly might make the complete system schedulable without changing the scheduler. By having such information, the designer can easily assess the impact of various alternatives to improve the performance, without having to change the implementation. It should be noted that an analysis of this type can't be done using methods like the RMS utilization test or reachability computation.

The algorithms described can be used to analyze the system in many different ways. For example, the effect of preemption on execution time can be assessed as follows. We have computed the maximum and minimum execution times for processes *after* they have been granted the CPU. If minimum and maximum are not the same, the process can be preempted after starting execution. For example, the display graphic subsystem can finish in as little as 7ms

and in as much as 14ms after it starts execution. In other words, preemption overhead can be as high as 7ms for this subsystem. The NAV steering subsystem has a minimum of 1ms and a maximum of 44ms. This means that other processes can delay it for 43ms. It is clear that NAV steering can be preempted for a longer time than display graphic, since it has lower priority. Our results, however, allow us to determine how much longer it can be preempted. In a similar fashion, we can compute the priority inversion time for high priority processes. This can aid in identifying the reasons why a system is not predictable, and help correct its behavior.

We examine one more property of this particular model. The weapon system is critical to the aircraft. It is very important that it respond quickly to the pilot's command. However, when a pilot presses the firing button, many subsystems are involved in identifying and responding to this event. By computing the minimum and maximum times between pressing the fire button and the execution of the weapon release process we are able to determine if the weapon system responds quickly enough to satisfy the aircraft requirements. In our example, the minimum time is 120ms and the maximum time is 167ms, not accounting for the possibility that the firing sequence may be aborted. Again, this type of analysis may be difficult to do with other tools. The RMS schedulability test cannot give tight bounds on specific response times for such properties, since its only parameter is CPU utilization. Algorithms that use reachability analysis are also inappropriate for such analysis. Specific exceptions, with previously defined time bounds, would have to be added to the model to observe these characteristics.

The finite-state model was implemented in about 600 lines of SMV code. The final model has about 10^{15} states, and the transition relation uses approximately 4600 BDD nodes. To compute each property described above took between 5 and 15 seconds using an i486 based workstation.

6 Conclusion

This paper proposes a general framework for computing quantitative characteristics of finite-state real-time systems. We have devised algorithms that calculate exact numerical bounds on the delay between two specified events, as well as on the frequency of the occurrence of a condition within a given interval. Rather than just determining the correctness of the model, the results computed by our algorithms provide hints about its behavior that can be useful in improving the performance of the system.

Our method can be easily integrated with model checking techniques. In fact, the *lower* and *upper bound* algorithms have been added to the most recent version of the SMV model checking system. Using this implementation we demonstrate the practical importance of our approach by analyzing a model of an aircraft control system. We have been able to obtain stronger results than those produced using traditional methods for real-time system verification.

We have found this approach to be very flexible. We have shown how quantitative characteristics can be computed for state-transition graphs. In addition, we have extended the algorithms to models in which transitions may take more than one time unit. We also plan to investigate the application of these techniques to other models of computation.

We believe that the quantitative information that our method provides can be extremely useful to designers during the development of real-time systems. We are confident that these techniques will prove practical in the verification of a variety of other realistic designs.

References

[1] R. Alur and T. A. Henzinger. Logics and models of real-time: a survey. In *Lecture Notes in Computer Science, Real-Time: Theory in Practice*. Springer-Verlag, 1992.

[2] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8), 1986.

[3] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *LICS*, 1990.

[4] S. V. Campos and E. M. Clarke. Real-time symbolic model checking for discrete time models. In *First AMAST International Workshop in Real-Time Systems*, 1993.

[5] S. V. Campos, E. M. Clarke, W. Marrero, M. Minea, and H. Hiraishi. Computing quantitative characteristics of finite-state real-time systems. Technical Report CMU-CS-94-147, Carnegie Mellon University, 1994.

[6] E. A. Emerson, A. K. Mok, A. P. Sistla, and J. Srinivasan. Quantitative temporal reasoning. In *Lecture Notes in Computer Science*. Springer–Verlag, 1990.

[7] A. N. Fredette and R. Cleaveland. RTSL: a language for real-time schedulability analysis. In *IEEE Real-Time Systems Symposium*, 1993.

[8] R. Gerber and I. Lee. A proof system for communicating shared resources. In *IEEE Real-Time Systems Symposium*, 1990.

[9] J. P. Lehoczky, L. Sha, J. K. Strosnider, and H. Tokuda. Fixed priority scheduling theory for hard real-time systems. In *Foundations of Real-Time Computing — Scheduling and Resource Management*. Kluwer Academic Publishers, 1991.

[10] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard real-time environment. *Journal of the ACM*, 20(1), 1973.

[11] C. D. Locke, D. R. Vogel, and T. J. Mesler. Building a predictable avionics platform in Ada: a case study. In *IEEE Real-Time Systems Symposium*, 1991.

[12] K. L. McMillan. *Symbolic model checking — an approach to the state explosion problem*. PhD thesis, SCS, Carnegie Mellon University, 1992.

[13] L. Sha, M. H. Klein, and J. B. Goodenough. Rate monotonic analysis for real-time systems. In *Foundations of Real-Time Computing — Scheduling and Resource Management*. Kluwer Academic Publishers, 1991.