Symbolic Schedulability Analysis of Task Sets with Arbitrary Deadlines

Mitchell L. Neilsen
Department of Computing and Information Sciences
Kansas State University
Manhattan, KS, USA

Abstract - *Times is a collection of tools designed for symbolic schedulability analysis and synthesis of code for real-time systems [1]. A design model in Times consists of a set of application tasks, a network of timed automata describing the task arrival patterns; e.g., periodic, sporadic, or controlled, and a real-time scheduling policy. Given a design model, Times will generate a scheduler, and calculate the worst-case response times for the tasks. The design model may be validated using the UPPAAL model checker [2, 3].*

However, if periodic task deadlines are allowed to be larger than their respective periods, then the worst-case response times that are reported by Times may be incorrect when using the default, two clocks scheduler. In particular, Times may report that a periodic task set is schedulable even though it is not. In this paper, we present an UPPAAL model with a two clocks scheduler that correctly computes the worst-case response times for periodic tasks with arbitrary deadlines.

Keywords: arbitrary deadlines, model checking, real-time scheduling, schedulability analysis, verification

1 Introduction

In classic real-time scheduling theory, tasks are frequently assumed to be periodic. To relax the traditional constraints on task arrival times, Times uses automata to model task arrival patterns [1]. The model is expressive enough to model real-time tasks that are periodic, sporadic, preemptive or non-preemptive, and tasks with additional precedence and resource constraints. Typical real-time scheduling algorithms can be easily generalized to automata. An automaton is schedulable if there exists a scheduling strategy such that all possible sequences of events accepted by the automaton are schedulable. It has been shown that the schedulability analysis problem for such models is decidable [4].

Schedulability analysis can be performed in a manner similar to response time analysis in classical real-time scheduling theory. Real-valued clocks can be used to model task execution and deadlines. The first main function of Times is developed based on these recent results on schedulability analysis. Its second main function is code generation. Code generation is to transform a validated design model to executable code whose execution preserves the behaviour of the model. Given a system design model in Times, including a set of application tasks, task constraints, tasks arrival patterns and a scheduling policy adopted on the target platform, Times will generate a scheduler and calculate the worst-case response times for all tasks. The model may be further validated by a model checker; e.g., using UPPAAL [2, 3]. Such model checkers can also be used to verify the correctness of other distributed algorithms [8, 9].

The focus of this paper is on extending the schedulability analysis in Times, but the results can be applied to the code generation which is already available in Times. For most cases, Times schedulability analysis is correct. However, if tasks in a periodic task set are allowed to have arbitrary deadlines – deadlines beyond task periods – then the worst-case response times reported by Times may be incorrect when using the default two clocks scheduler. Consequently, the schedulability analysis tool may report that a task set is schedulable when it is not. In addition, Times may also report incorrect worst-case response times when the task set is feasible. It is important to note that the two clocks scheduler presented in [5, 6] is correct, but the implementation of the two clocks scheduler in Times has some flaws.

The rest of the paper is organized as follows: the next section describes the input language for Times and UPPAAL. Section 3 summarizes schedulability analysis in Times, and provides a counter-example that clearly illustrates the problem with the existing schedulability analysis tool in Times. Section 4 describes a new, simple two clocks model that can be used to correctly compute the worst-case response times for this counter-example, and correctly test for schedulability using UPPAAL queries. Section 5 concludes the paper with a brief summary.

2 Times Task Model

In Times, a task (or task type) is an executable program with several task parameters including worst-case execution time and deadline. A task model specifies a task arrival pattern such as periodic. Timed automata can be used to describe several different task arrival patterns. The focus of this paper is on periodic task sets. Each task τ_i is characterized by a pair of natural numbers denoted (C_i, D_i) with $C_i \leq D_i$, where C_i is the run time of task τ_i and D_i is its relative deadline; that is, after task τ_i is released for execution, it should complete within D_i time units. For periodic tasks, the period is denoted by T_i. In Times, the inputs for C_i, D_i, and T_i are labeled C, D, and T, respectively, as shown below in Figure 1.

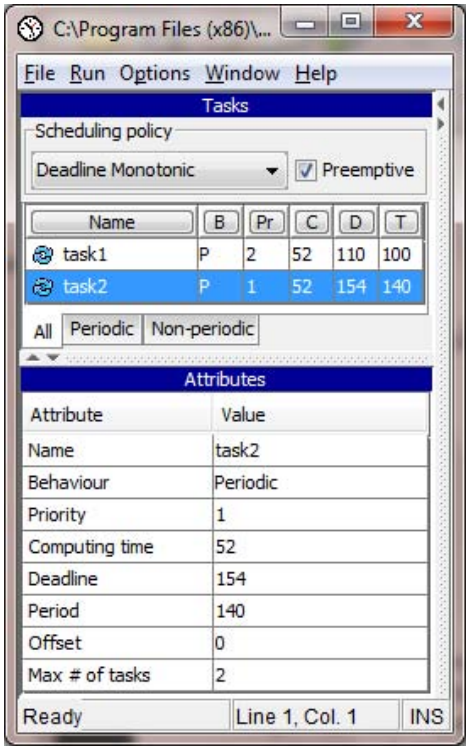

Figure 1. Periodic task set

This task set is the second example from Lehoczky's paper [7]. This example was used to show that a Deadline Monotonic (DM) priority assignment may not be optimal for task sets with arbitrary deadlines; that is, tasks with deadlines greater than their periods. Note that in this case, two jobs in task2 (τ_2) may be ready to execute at the same time because the second job is released at time 140 and the first job, which has a deadline of 154, will still be executing at time 140. Thus, the input to Times must specify an upper bound on the number of jobs that can be in the ready queue or running. An absolute upper bound is $\lceil D_i/C_i \rceil$, as discussed in [5], but for this particular task set, we can specify an upper bound (specified as the "Max # of tasks" field in Figure 1) of 2; that is, $\lceil D_i/T_i \rceil$.

The core of the Times input language is timed automata extended with data variables and tasks. As in an UPPAAL model, each edge of such an extended automaton can be labeled with three labels:

1. a guard containing a clock constraint and/or a predicate on data variables,
2. an action which can be an input or output action in the form of a! or a?, and
3. a sequence of assignments in the form: x = 0 when x is a real-valued clock or v = E when v is a data variable and E is a mathematical expression over data variables and constants.

A location of an extended automaton may be annotated with a task or a set of tasks that will be triggered when the transition leading to the location is taken. The triggered tasks will be put into a task queue, like the ready queue in an operating system, and scheduled to run according to a given scheduling policy. The scheduler should make sure that all task constraints are satisfied in scheduling the tasks in the task queue. To model concurrency and synchronization between automata, networks of automata are constructed in the standard way as in UPPAAL with the annotated sets of tasks on locations unioned [1, 6].

Four types of shared data variables can be used for communication and resource sharing:

1. Tasks can share variables with each others, protected by semaphores.
2. Tasks can read and update variables owned by the automata.
3. Automata can read (but not update) variables owned by the tasks.
4. Automata can share variables with each other.

In this paper, we limit the focus to periodic tasks that are independent and do not share resources. The next section describes how schedulability analysis is performed in Times.

3 Schedulability Analysis

In Times, a network of timed automaton annotated with tasks is considered as a design model.

Given an extended automaton and a scheduling policy, the related schedulability analysis problem is to check whether there exists a reachable state of the scheduler automaton where a task misses its deadline. Such states are called non-schedulable states. An automaton is said to be non-schedulable with the given scheduling policy if it may reach a non-schedulable (ERROR) state. Otherwise, the automaton is schedulable.

Consider the task set shown in Figure 1. Both tasks have run times of 52. The high priority task (priority Pr = 2) has a deadline of 110 and a period of 100. The low priority task (priority Pr = 1) has a deadline of 154 and a period of 140.

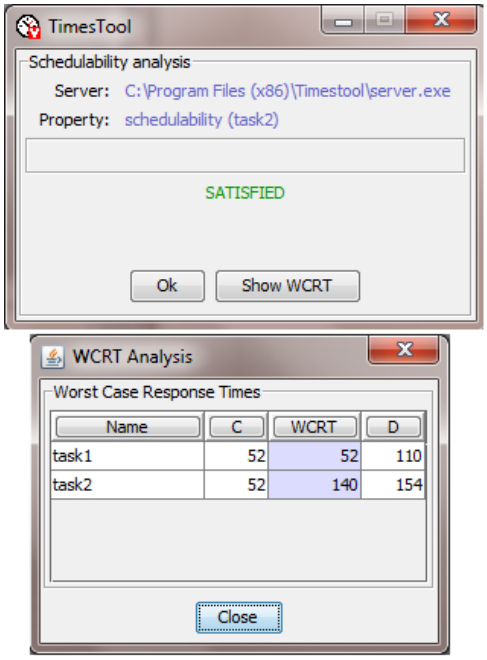

Figure 2. Times output

Even though the task set is not schedulable, the Schedulability Analysis Tool in Times, using the default settings, reports that the task set is feasible as shown in Figure 2. Times also reports that the worst-case response time for task 2 is 140, which is the period of task 2, but this is incorrect.

It is interesting to note that this error only occurs when using the two clocks scheduler, which is the default option as shown in Figure 3.

If you uncheck this selection under the Options menu, then a more general scheduler that uses more clocks correctly reports that the task set is not schedulable.

To check if a task set is schedulable, Times constructs a pair of timed automata – one to generate jobs to be released (called the task automaton) and one to schedule the jobs released (called the scheduler automaton). Then, Times checks the reachability of a predefined error state in the product automaton of the pair. If the error state is reachable, the task automaton is not schedulable with the scheduler automaton.

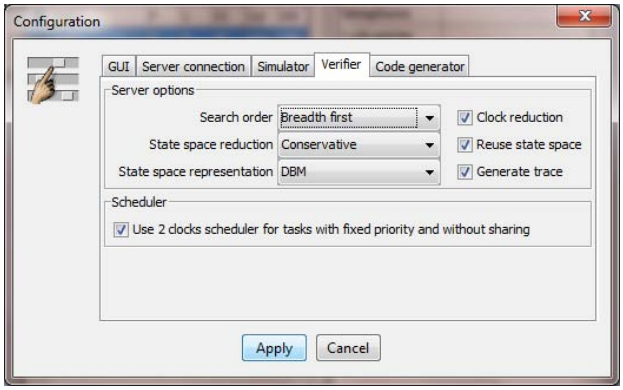

Figure 3. Default options

In this paper, we will focus on two clocks scheduler automata. The encoding of a two clocks scheduler is used to test the feasibility of task τ_i. As in the encoding of a two clocks scheduler by Fersman, et al. [6], there are four states:

- $Idle_i$ - the ready job queue is empty,
- $Busy_i$ - jobs with priority greater than or equal to task i have arrived for execution,
- $Check_i$ - the job in task i to be checked for feasibility is executing, and
- $Error_i$ - the checked job in task i missed its deadline.

The new, two clocks model uses a model that includes fewer transitions and fewer variables than the two clocks model used in Times. However, the more general simulator model in Times which can be exported uses clock differences; that is, when a job completes, the clocks are updated based on clock differences. Unfortunately, this is not allowed in the current version (version 4.0.10, September 2009) of UPPAAL. Also, the two clocks version of the scheduler cannot be exported from Times, so we were not able to identify the exact error that caused the two clocks version to fail in Times.

In the next section, we introduce a new two clocks model that can be used to test the feasibility of periodic task sets with arbitrary deadlines.

4 New Two Clocks Model

As in Times, the first automaton is designed to model the releases of jobs within each task. The automaton for the example given above is shown above

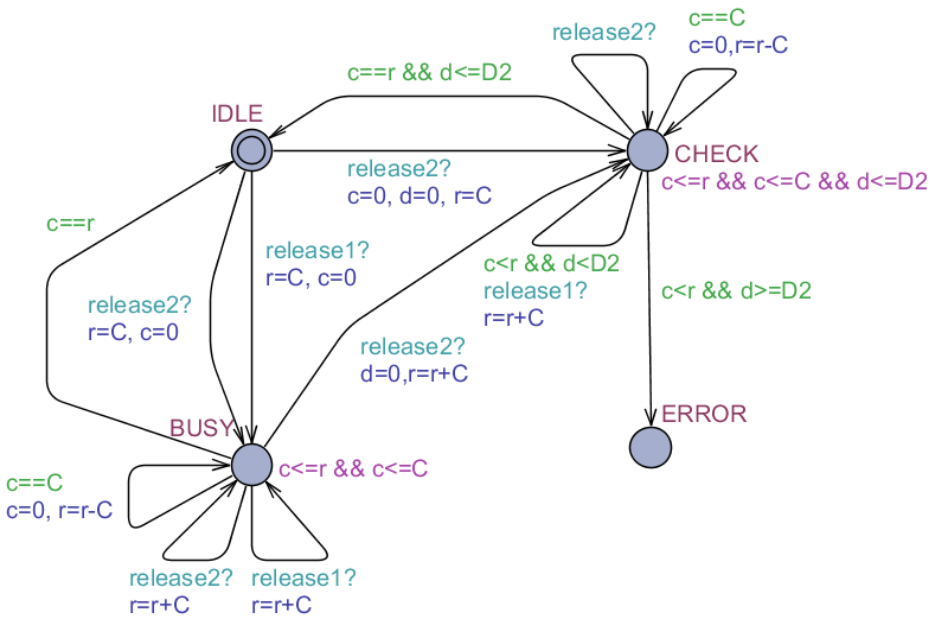

Figure 4. SCHEDULER automaton

in Figure 4. The following global declarations are used to specify the same constraints shown in Figure 1:

```
const int C = 52;
const int D1 = 110;
const int D2 = 154;
const int T1 = 100;
const int T2 = 140;
```

As in Times, global channels, release1 and release2, are used by the PERIODIC_TASKS automaton to tell the SCHEDULER automaton that a new job from task τ_1 or task τ_2 has been released.

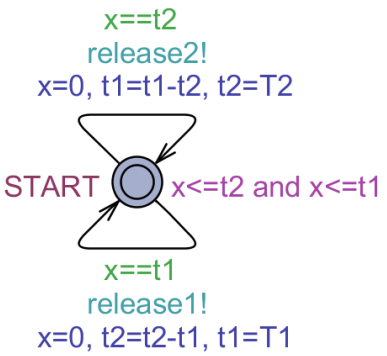

Figure 5. PERIODIC_TASKS automaton

A local clock, x, is used to determine when the jobs are released. The PERIODIC_TASKS automaton, shown in Figure 5, is essentially the same as the job generating automaton used to generate periodic jobs in Times. The main differences are in the SCHEDULER automaton shown in Figure 4. The intuition is the same as that used for traditional real-time response time analysis. In this model, the high priority task is designated as task τ_1, and the low priority task is designated as task τ_2. When a message is received on channel release1, a job has been released in task τ_1. Some job in task τ_2 is non-deterministically selected to be checked for feasibility. The model checker will check all possible jobs in task τ_2. The intuition behind the model is that if some job will miss its deadline, then all jobs in task τ_2 that meet their deadlines will be processed in the BUSY state. Then, when the job that will miss its deadline is released, a transition is taken to the CHECK state. When the deadline is reached before the job has finished executing, a transition is taken to the ERROR state. If the first job will miss its deadline, then a transition can be taken directly from the IDLE state to the CHECK state. We rely on the model checker to test all possible jobs in the low priority task to determine if there is a job that will miss its deadline.

Using the UPPAAL Verification Tool, a user can quickly check to see if the SCHEDULER ever enters the ERROR state by using the property: E<>(SCHEDULER.ERROR). If the propery is satisfied then, on some path, the SCHEDULER automaton eventually enters the ERROR state indicating that the task set is not schedulable. For the sample data given, using a deadline monotonic priority assignment, this property is satisfied. If some job misses its deadline, a trace can be generated and visualized in UP-

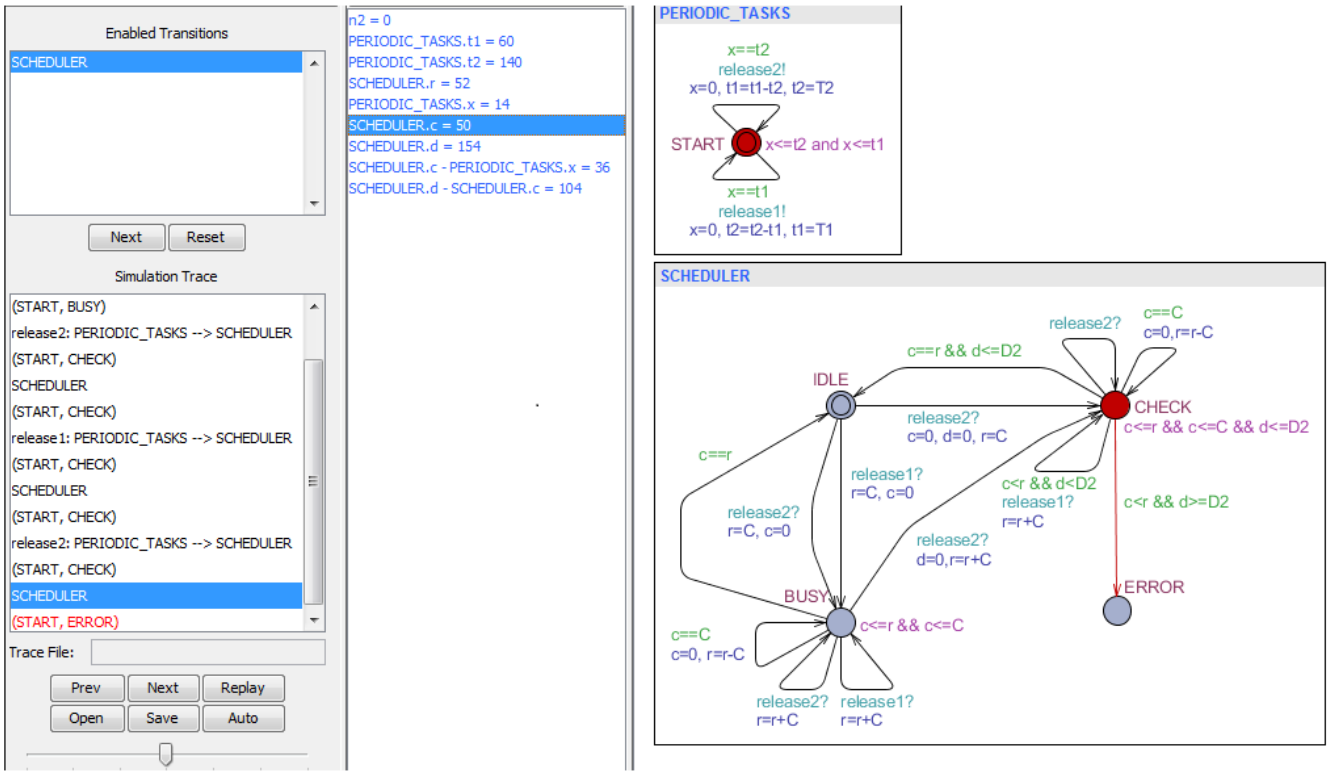

Figure 6. Missed deadline

PAAL leading to the simulator output shown in Figure 6. When the SCHEDULER automaton goes from the CHECK state to the ERROR state, the value of c = 50, but r = 52 and d=154; that is, the first job in task τ_2 has run for 50 of 52 time units needed to complete its execution when it misses its deadline at time 154 (D2 = 154).

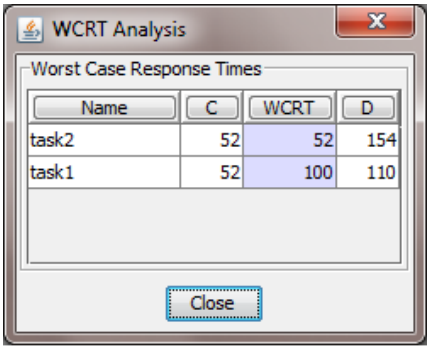

Figure 7. Times output

If we switch the priorities of the tasks shown in the example, then the task set is schedulable. In this case, task τ_2 the low priority task, is set to have a deadline of 110 and a period of 100.
This can be easily done in the model by simply resetting the global constants; e.g., swapping the values assigned to D1 and D2, and swapping the values assigned to T1 and T2.

```
const int D1 = 154;
const int D2 = 110;
const int T1 = 140;
const int T2 = 100;
```

Using Times, the worst-case response time for task τ_2 is reported to be 100, as shown in Figure 7, but this is the period for task τ_2, and is incorrect. Using the UPPAAL Verification Tool, it is possible to compute the worst-case response time for each task using the property: A[](SCHEDULER.CHECK imply (SCHEDULER.d<=108)) which is satisfied. The same property with an upper bound of 107 is not satisfied, as shown in Figure 9. Thus, the worst-case response time of task τ_2 is 108, which is less than it's deadline of 110.

In addition to scheduling properties, it is possible analyze other safety and liveness properties using UPPAAL. In particular, it is easy to show that the clock c is bounded above by 124, and the integer variable r is bounded by 104, as shown in Figure 9.

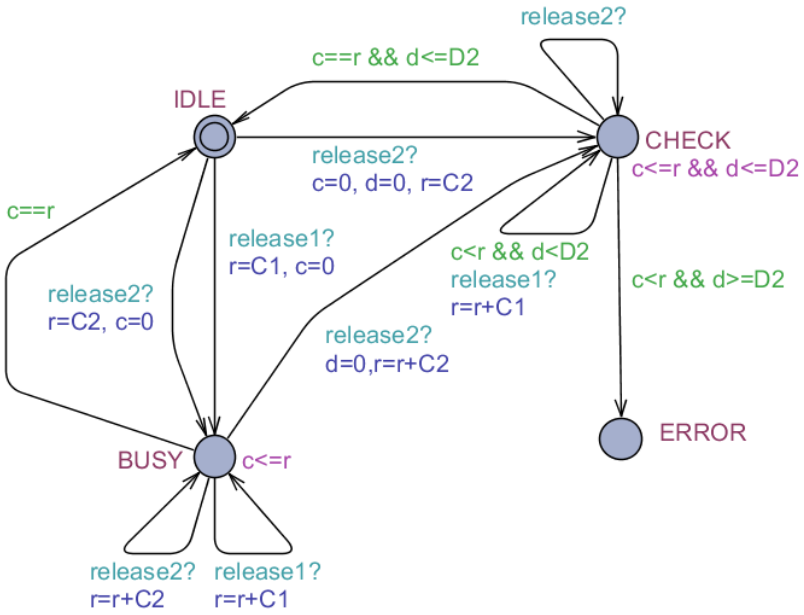

Figure 8. Updated SCHEDULER automaton

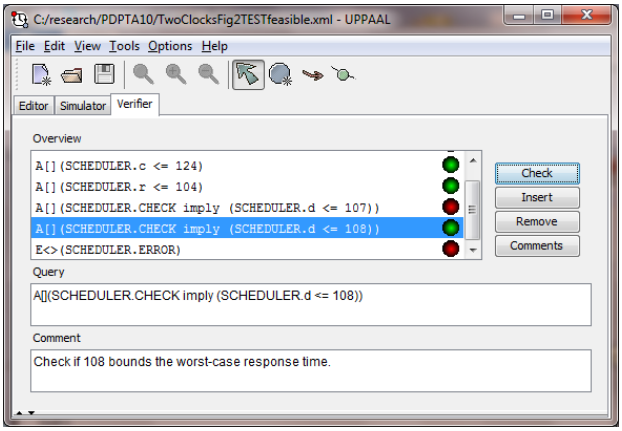

Figure 9. Verification output

To model the third example from Lehoczky's paper [7], we need to update the SCHEDULER automaton with two clock values, C1 = 26 for the run-time of jobs in task τ_1 and C2 = 62 for the run-time of jobs in task τ_2, and let C denote the maximum of C1 and C2:

```
const int C1 = 26;
const int C2 = 62;
const int D1 = 70;
const int D2 = 118;
const int T1 = 70;
const int T2 = 100;
```

A few transitions are updated to reflect changes in r based on which task released the job, as shown in Figure 8. In addition, transitions used to bound r are removed; in particular, a trinsition from BUSY and a transition from CHECK that decrements r by C is removed. As noted in [**?**], the value used for C to decrement r can be any positive value. It turns out that if C is larger than the level-i busy period, then the number of states generated and searched will be the least possible, but the value of r may reach the length of the level-i busy period. For this problem, the number of states searched for each value is shown below:

C	**States**	**C**	**States**
1	3337	100	174
3	1322	125	141
6	729	250	129
12	419	694	123
50	212	695	120

If a message is received on channel request1, denoted request1?, then a job has been released in task τ_1. So, the value of r should be updated by adding C1, the run-time of jobs in task τ_1. On the other hand, if a message is received on channel request2, denoted request2?, then a job has been released in task τ_2, and the value of r should be updated by adding C2.

Queries can be used to verify that the task set is feasible and that the worst-case response time is 118 as shown in Figure 10. In particular, the query E<>(SCHEDULER.ERROR) is not satisfied, proving that the task set is feasible. The property A[](SCHEDULER.CHECK imply (SCHEDULER.d<=118)) is satisfied, proving that the

worst-case response time of task τ_2 is bounded by 118, and is indeed the worst-case response time based on the next query.

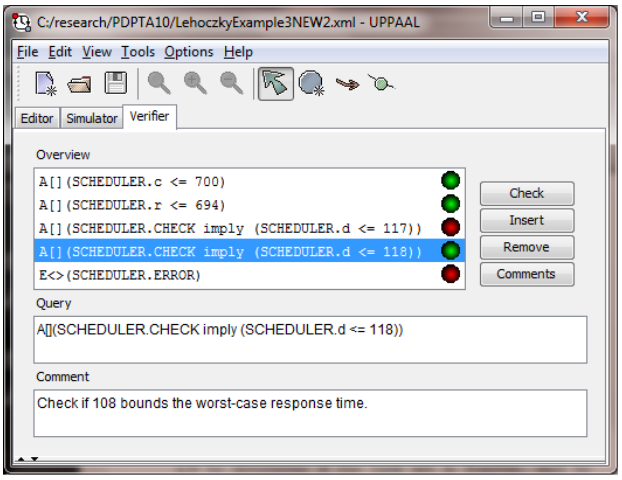

Figure 10. Verifier output

The verifier can also be used to show that the clock c is bounded above by 700, and the integer variable r is bounded by 694, as shown in Figure 10. In this case, the level-i busy period is [0,694], even though it is incorrectly noted as [0,696] in Lehoczky's paper [7]. So, in this way, the verifier can be used to determine the length of the level-i busy period, and the clock c is bounded by the least common multiple of the task periods.

5 Conclusions

This paper presented a simple model that can be used to analyze the feasibility of periodic task sets with arbitrary deadlines. The model consists of a set of two automata: a PERIODIC_TASKS automaton to model the release of periodic jobs, and a SCHEDULER automaton to model the scheduler. The model generated can be used in conjunction with UPPAAL 4.0 to determine if the task set is feasible, and to compute the worst-case response times of each task in the system. If the task set is not feasible, the model can also be used to identify why the task set fails to meet its deadlines. The example UPPAAL models and queries generated are available on-line at: `http://www.cis.ksu.edu/~neilsen/pdpta10/`.

The model and queries could easily be generalized to test the feasibility of other periodic task sets with arbitrary start times and arbitrary deadlines. In particular, we didn't even check to see if the high priority tasks are schedulable, but this could be modeled very easily. UPPAAL is a very powerful tool that can be used for many types of verification for distributed and real-time systems.

References

[1] T. Amnell, E. Fersman, L. Mokrushin, P. Pettersson, and W. Yi. Times: a tool for schedulability analysis and code generation of real-time systems. In *In Proc. of FORMATS03, number 2791 in LNCS*, pages 60–72. Springer-Verlag, 2003.

[2] G. Behrmann, A. David, K.G. Larsen, J. Håkansson, P. Pettersson, W. Yi, and M. Hendriks. UPPAAL 4.0. In *Proceedings of the 3rd International Conference on the Quantitative Evaluation of SysTems (QEST) 2006*, IEEE Computer Society, pages 125–126, 2006.

[3] J. Bengtsson, F. Larsson, P. Pettersson, W. Yi, P. Christensen, J. Jensen, P. Jensen, K. Larsen, and T. Sorensen. Uppaal: a tool suite for validation and verification of real-time systems, 1996.

[4] E. Fersman, P. Pettersson, and W. Yi. Timed automata with asynchronous processes: schedulability and decidability. In *In Proceedings of TACAS 2002*, pages 67–82. Springer-Verlag, 2002.

[5] Elena Fersman, Leonid Mokrushin, Paul Pettersson, and Wang Yi. Schedulability analysis using two clocks. In *In 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS*, pages 224–239. Springer, 2003.

[6] Elena Fersman, Leonid Mokrushin, Paul Pettersson, and Wang Yi. Schedulability analysis of fixed-priority systems using timed automata. *Theoretical Computer Science*, 354(2):301 – 317, 2006. Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2003).

[7] J.P. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *IEEE Real-Time Systems Symposium*, pages 201–213, 1990.

[8] M.L. Neilsen. A generalized token-based mutual exclusion algorithm for wireless networks. In *Proceedings of the 20th Int'l Conference on Parallel and Distributed Computing Systems (PDCS-2007), Las Vegas, Nevada, USA*, ISCA, 2007.

[9] M.L. Neilsen. Model checking token-based distributed mutual exclusion algorithms. In *Proceedings of the 15th International Conference on Parallel and Distributed Processing Techniques and Applications*, pages 10–16, 2009.