Schedulability of Herschel-Planck Revisited Using Statistical Model Checking *

Alexandre David[1], Kim G. Larsen[1], Axel Legay[2], Marius Mikučionis[1]

[1] Computer Science, Aalborg University, Denmark
[2] INRIA/IRISA, Rennes Cedex, France

Abstract. Schedulability analysis is a main concern for several embedded applications due to their safety-critical nature. The classical method of response time analysis provides an efficient technique used in industrial practice. However, the method is based on conservative assumptions related to execution and blocking times of tasks. Consequently, the method may falsely declare deadline violations that will never occur during execution. This paper is a continuation of previous work of the authors in applying extended timed automata model checking (using the tool UPPAAL) to obtain more exact schedulability analysis, here in the presence of non-deterministic computation times of tasks given by intervals [BCET,WCET]. Considering computation *intervals* makes the schedulability of the resulting task model undecidable. Our contribution is to propose a combination of model checking techniques to obtain some guarantee on the (un)schedulability of the model even in the presence of undecidability.
Two methods are considered: symbolic model checking and statistical model checking. Symbolic model checking allows to conclude schedulability – i.e. absence of deadline violations – for varying sizes of BCET. However, the symbolic model checking technique is over-approximating for the considered task model and can therefore not be used for *dis*proving schedulability. As a remedy, we show how statistical model checking may be used to generate concrete counter examples witnessing non-schedulability. In addition, we apply statistical model checking to obtain more informative performance analysis – e.g. expected response times – when the system *is* schedulable.
The methods are demonstrated on a complex satellite software system yielding new insights useful for the company.

1 Introduction

Embedded systems involve the monitoring and control of complex physical processes using applications running on dedicated execution platforms in a resource constrained manner in terms of for example memory, processing power, bandwidth, energy consumption, as well as timing behavior.

* Work partially supported by VKR Centre of Excellence MT-LAB, the Sino-Danish basic research center IDEA4CPS, the regional CREATIVE project ESTASE, and the EU projects DANSE and DALI.

Viewing the application as a collection of (interdependent tasks) various scheduling principles may be applied to coordinate the execution of tasks in order to ensure orderly and efficient usage of resources. Based on the physical process to be controlled, timing deadlines may be required for the individual tasks as well as the overall system. The challenge of schedulability analysis is now concerned with guaranteeing that the applied scheduling principle(s) ensure that the timing deadlines are met.

The classical method of response time analysis [JP86,Bur94] provides an efficient means for schedulability analysis used in industrial practice: by calculating safe upper bounds on the worst case response times (WCRT) of tasks (as solutions to simple recursive equations) schedulability may be concluded by a simple comparison with the deadlines of tasks. However, classical response time analysis is only applicable to restricted types of task-sets (e.g. periodic arrival patterns), and only applicable to applications executing on single processors. Moreover, the method is based on conservative assumptions related to execution and blocking times of tasks [BHK99]. Consequently, the method may falsely declare deadline violations that will never occur during execution.

Process algebraic approaches [BACC$^+$98,SLC06] have resulted in many methods for specification and schedulability analysis of real-time systems. Timed automata frameworks are also combined with other tools [BHK$^+$04,BHM09] for schedulability and schedule assessment in realistic settings.

In this paper, we continue our effort in applying real-time model checking using the tool UPPAAL to obtain more exact schedulability analysis for a wider class of systems, e.g. task-sets with complex and interdependent arrival patterns of task, multiprocessor platforms, etc [DILS10]. In particular, we revisit the industrial case study of the Herschel-Planck [MLR$^+$10] satellite system. The control software of this system – developed by the Danish company Terma A/S – consists of 32 tasks executing on a single processor system with preemptive scheduling, and with access to shared resources managed by a combination of priority inheritance and priority ceiling protocols. Unfortunately, though falling into the class of systems covered by the classical response time analysis, this method fails to conclude schedulability for the Herschel-Planck application.

In our previous work [MLR$^+$10], we "successfully" concluded schedulability of the Herschel-Planck application using a more exact analysis based on the timed automata modeling framework for schedulability problems of [DILS10] and the tool UPPAAL. However, the analysis in [MLR$^+$10] is based on the strong *assumption*, that each task has a given and specific execution time (ET). Clearly this is an unrealistic assumption. In reality, it can only be guaranteed that the execution time for (each execution of) task i is in some interval [BCET_i,WCET_i]. In particular, classical response time analysis aims at guaranteeing schedulability for intervals [0,WCET_i], though unfortunately inconclusive for the Herschel-Planck control software. On the other hand the guarantee of schedulability in [MLR$^+$10] only applies to (unrealistic) singleton intervals [WCET_i,WCET_i].

Both works consider systems with preemptive scheduling and our timed automata models are using stop-watches. In addition this work deals with non-

$f = \frac{\text{BCET}}{\text{WCET}}$:	0-71%	72-86%	87-90%	90-100%
Symbolic MC:	maybe	maybe	n/a	**Safe**
Statistical MC:	**Unsafe**	maybe	maybe	maybe

Table 1: Summary of schedulability of Herschel-Planck concluded using symbolic and statistical model checking.

deterministic computation times as well as dependencies between task due to resources, thus schedulability of the resulting task model is undecidable [FKPY07]. Our solution will be to consider a combination of well-known model checking techniques to obtain some guarantee on schedulability even in the presence of undecidability. Concretely, we revisit the schedulability problem for the Herschel-Planck software, with execution time intervals of the general form [BCET_i,WCET_i], with $\text{BCET}_i \leq \text{WCET}_i$. Two model checking methods of Uppaal are applied for the schedulability analysis: classical (zone-based) symbolic model checking (MC) and statistical model checking (SMC). The core idea of SMC [LDB10,YS06,SVA04,KZH+09] is to randomly generate simulations of the system and verify whether they satisfy a given property. The results are then used by statistical algorithms in order to compute among others an estimate of the probability for the system to satisfy the property. Such estimate is correct up to some confidence that can be parameterized by the user. Several SMC algorithms that exploit a stochastic semantics for timed automata have recently been implemented in Uppaal [DLL+11a,DLL+11b,BDL+12].

Symbolic MC allows to conclude schedulability – i.e. absence of deadline violations – for varying sizes of BCET, though with the size of the (symbolic) state-space and the overall verification time increasing significantly with an increase in the size of the intervals [BCET_i,WCET_i]. Moreover, the symbolic MC technique is over-approximate due to the presence of stop-watches needed to encode preemption. Thus, symbolic MC can not be used for *dis*proving schedulability. As a remedy, we show how statistical MC may be used to generate concrete counter examples witnessing non-schedulability. The new results obtained for Herschel-Planck are rather interesting as can be seen from the summary Table 1: when $\frac{\text{BCET}}{\text{WCET}} \geq 90\%$ symbolic MC confirms schedulability, whereas statistical MC disproved schedulability for $\frac{\text{BCET}}{\text{WCET}} \leq 71\%$. For $\frac{\text{BCET}}{\text{WCET}} \in (71\%, 90\%)$ both methods are inconclusive either due to the over-approximation induced by the symbolic approach or to a burden in computation time. In addition, we apply statistical MC to obtain more informative performance analysis, e.g. expected response times when the system is schedulable as well as estimation of the probability of deadline violation when the system is not schedulable.

2 Modeling

This section introduces the modeling framework that will be used through the rest of the paper. We start by presenting the generic features of the framework through a running example. Then, we briefly indicate the key additions in modeling the Herschel-Planck application, leaving details to be found in [MLR+10].

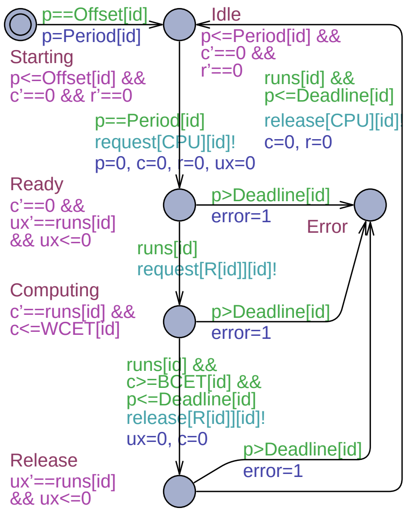
We consider a *Running Example*, that builds on instances from a library of three types of processes represented with timed automata templates in UPPAAL: (1) preemptive CPU scheduler, (2) resource schedulers that can use either priority inheritance, or priority ceiling protocols, and (3) periodically schedulable tasks. In what follows, we use broadcast channels in entire model which means that the sender cannot be blocked and the receiver can ignore it if it is not ready to receive it. Derivative notation like `x'==e` specifies whether the stop-watch `x` is running, where `e` is an expression evaluating to either `0` or `1`. By default all clocks are considered running, i.e. the derivative is `1`.

For simplicity, we assume periodic tasks arriving with period `Period[id]`, with initial offset `Offset[id]`, requesting a resource `R[id]`, executing for at least best case execution time `BCET[id]` and at most worst case execution time `WCET[id]` and hopefully finishing before the deadline `Deadline[id]`, where `id` is a task identifier. The wall-clock time duration between task arrival and finishing is called response time, it includes the CPU execution time as well as any preemption or blocking time.

Figure 1a shows the UPPAAL declaration of the above mentioned parameters for three tasks (number of tasks is encoded with constant `NRTASK`) and one resource (number of resources is encoded with constant `NRRES`). The `task_t` type declaration says that task identifiers are integers ranging from 1 to `NRTASK`. Similarly the `res_t` type declares resource identifier range from 1 to `NRRES`. Parameters `Period`, `Offset`, `WCET`, `BCET`, `Deadline` and `R` are represented with integer arrays, one element per each task. Figure 1b shows a simple periodic task template which starts in `Starting` location, waits for the initial offset to elapse and then moves to `Idle` location. The task arrival is marked by a transition to a `Ready` location which signals the CPU scheduler that it is ready for execution. Then location `Computing` corresponds to busy computing while holding the resource and `Release` is about releasing the resources and finishing. The periodicity of a task is controlled by constraints on a local clock `p`: the task can move from `Idle` to `Ready` only when `p==Period[id]` and then `p` is reset to zero to mark the beginning of a new period. Upon arrival to `Ready`, other clocks are also reset to zero: `c` starts counting the execution time, `r` measures response time and `ux` is used to force the task progress. The invariant on location `Ready` says that the task execution clock `c` does not progress (`c'==0`) and it cannot stay longer than zero time units (`ux<=0`) unless it is not running (`ux'==runs[id]`). The task also cannot progress to location `Computing` unless the CPU is assigned to it (`runs[id]` becomes true). When the CPU is assigned, the task will be forced to urgently request a resource and move on to `Computing`, where the computation time (valuation of `c`) increases only when it is marked as running (`runs[id]` is true). The task can stay in `Computing` for at most worst case execution time (`c<=WCET[id]`), cannot leave before best case execution time (`c>=BCET[id]`), but can be preempted by setting `runs[id]` to `0`. If the resource is not granted then the resource scheduler is responsible for blocking the task from using the

```
const int NRTASK = 3; // # of tasks
const int NRRES = 1; // # of resources
typedef int [1,  NRTASK] task_t;
typedef int [1,  NRRES] res_t;
const int f=80; // fraction of WCET, in %
int Period[task_t]    = {  100,   100,  100 };
int Offset[task_t]    = {   20,     0,   10 };
int WCET[task_t]      = {   15,    25,   40 };
int BCET[task_t]      = { WCET[1]*f/100,
     WCET[1]*f/100, WCET[1]*f/100 };
int Deadline[task_t]  = {   20,    40,   70 };
res_t   R[task_t]     = {    1,     1,    1 };
int P[task_t]  = { 3, 2, 1 }; // priorities
bool runs[task_t]  = { 0, 0, 0 };
bool error  = false; // global variable
```

(a) Task parameters.



(b) Task template.

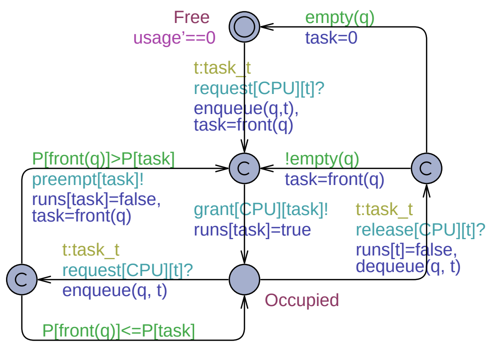
Fig. 1: Task parameter declaration and its stop-watch automaton template.

CPU. If the deadline is not violated (`p<=Deadline[id]`) then the task can move on to `Release` and similarly complete to `Idle`. Notice that task competes for resources in locations `Ready`, `Computing` and `Release` and it will move to `Error` location if the deadline is exceeded (`p>Deadline[id]`).

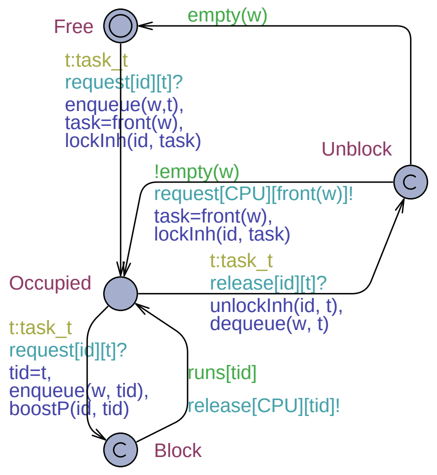
The CPU scheduler is equipped with a task queue `q` sorted by task priorities `P[t]`, where `t` is a task identifier and `task` variable holding the currently running task identifier. Function `front(q)` always returns the highest priority task identifier in the `q` queue. Figure 2a shows a CPU template which alternates between `Free` and `Occupied` locations.

When a `request[CPU][t]` arrives, the requesting task `t` is put into the queue and the CPU is being rescheduled. This is done either by immediate `grant[CPU][task]` and marking that the task is running `runs[task]=true`, or via preemption of the currently running task of lower priority, or simply returning to `Occupied` if the highest priority task in the queue is not higher than currently running. When a `release[CPU][t]` arrives, the requesting task `t` is de-queued, marked as not running (`runs[t]=false`), and the CPU is granted to the next highest priority task in the queue (if the queue is not empty). We use Uppaal committed locations (encircled with `C`) for uninterrupted (atomic) transitions, thus `Free` and `Occupied` are the only locations where the time can pass. In addition, the scheduler is equipped with `usage` stop-watch: `usage` is stopped by invariant `usage'==0` at location `Free` and is running with default rate of `1` in location `Occupied`, hence its valuation computes the CPU usage.

A resource scheduler shown in Fig. 2b is equipped with its own waiting queue `w`. It operates in a similar way as CPU scheduler, that is by alternating between



(a) Preemptive CPU scheduler.



(b) Resource using priority inheritance.

Fig. 2: Schedulers for active and passive resources.

`Free` and `Occupied`. The main difference is that a resource cannot be preempted once it is locked. The locking operations follow the priority inheritance protocol implemented in functions `lockInh(res,task)`, `unlockInh(res,task)`. Operation `boostP(res,task)` raises the priority of the resource `res` owner to higher level than the requesting `task`. Figure 3 shows the listing of Uppaal code implementing the priority inheritance protocol.

```
/** Boost the priority of resource owner based on priority inheritance protocol: */
void boostP(res_t res, task_t task) {
    if (P[owner[res]] <= defaultP(task)) {
        P[owner[res]] = defaultP(task)+1;
        sort(q); // sorts the queue by descending priorities
    }
}
/** Lock the resource based on priority inheritance protocol: */
void lockInh(res_t res, task_t task) {
    owner[res] = task; // mark the resource as occupied by the task
}
/** Unlock the resource based on priority inheritance protocol: */
void unlockInh(res_t res, task_t task) {
    owner[res] = 0; // mark the resource as released
    P[task] = defaultP(task); // return to default priority
}
```

Fig. 3: Data and function declarations

Similarly to priority inheritance scheduler, we can also model priority ceiling protocol by suitable modification of the locking functions.

Herschel. In this paper, we will also consider a large, industrial case study: the schedulability analysis of the control software of the Herschel satellite. This case study, which will seriously challenge the capabilities of Uppaal, uses the

Table 2: Summary of schedulability of the *Running Example* example concluded using symbolic and statistical MC for varying sizes of computation time intervals.

$f = \frac{BCET}{WCET}$	0-79%	80-83%	84-100%
Symbolic MC:	maybe	maybe	**Safe**
Statistical MC:	**Unsafe**	maybe	maybe

same basic stop-watch modeling principles as in the *Running Example* described above. The Herschel model consists of 32 tasks sharing 6 resources using two protocols (priority inheritance and priority ceiling). Among these tasks, 24 are periodic [3], while 8 are triggered in a sequence. Additionally, tasks use multiple resources in a sequence, thus the sequence between `Idle` and `Release` of acquiring and releasing resources is more refined. As an optimization, the resource sharing and blocking is built into tasks to alleviate the need for task queues. We refer the reader to [MLR$^+$10] for more details.

3 Symbolic Safety Analysis

In this section we apply the classical zone-based symbolic reachability engine of Uppaal to verify schedulability. As we are considering systems with preemptive scheduling our models will be using stop-watches. With the addition of having non-deterministic computation times – i.e. computation *intervals* – as well as dependencies between task due to resources, schedulability of the resulting task model is undecidable as a consequence of the results of [FKPY07]. In this case, the analysis provided by Uppaal is over-approximate, guaranteeing that safety properties established are valid properties of the system but leaving reachability properties to be possibly spurious.

Running Example As detailed in Section 2, our running example consists of three tasks with WCET times being $15, 25, 40$, deadlines $20, 40, 70$ and with one single resource shared by the three tasks. In Table 2 the result of applying symbolic MC with respect to the safety property

```
A[]  !error
```

is given. Here `error` is a Boolean being set whenever a task misses its deadline (see Figure 1b). Thus this property expresses absence of deadline violations (i.e. schedulability), and is confirmed (within 0.06s) for computation intervals $[f \cdot \mathrm{WCET}, \mathrm{WCET}]$ with $f \geq 84\%$. For $f < 84\%$, the over-approximate analysis of Uppaal returns symbolic counter-example traces indicating possible deadline violations for task `T(1)`. However, these may be spurious.

In addition to schedulability, we may obtain upper bounds on the WCRTs of the three tasks by posing the the query

[3] In actual system 16 of them are sporadic, but we model them as periodic to limit non-determinism.

```
sup:  T(1).r, T(2).r,  T(3).r
```

where `T(i).r` is a stopwatch running whenever the task `T(i)` is not idle. Again the results fall in two classes: for computation intervals $[f \cdot \mathrm{WCET}, \mathrm{WCET}]$ with $f \geq 84\%$ the WCRTs are $20, 40, 70$ and for $f < 84\%$ the WCRTs are $55, 40, 70$, again indicating the possibility of deadline violation for task `T(1)`.

Finally, using an additional stopwatch `usage`, which is only stopped when the CPU is free (and reset for each 2000 time-units) the query `sup: usage` returns the value 1600, providing 80% (= 1600/2000) as an upper bound of the CPU utilization.

Herschel. Applying in a similar manner symbolic MC to the Herschel case seriously challenges the engine of UPPAAL, due to the the explosion in the size symbolic state-space with the increase of the size of the computation time intervals. In fact, to avoid the analysis to run out of memory we have applied the so-called sweep-line method.

Table 3 provides a summary of the effort spent in verifying the model. We started verification with model-time limited instances to get an impression of resources need to verify the model and once we gained enough confidence we increased the limit, thus the results are sorted by the model-time limit. The deterministic case of $f = 100\%$ is relatively cheap and even unlimited case is verifiable within three hours. Important insight here is that the verification time correlates linearly with the limit and the unlimited case seems to correlate with 156 cycles which is the least common multiple of all task periods. So given enough time we managed to verify down to $f = 90\%$ where the resource consumption is increased drastically to more than 6 days. Finally, the model-checker indicate a (possibly spurious) deadline violation for the case $f = 86\%$ after a little bit more than 4 days.

Table 3: Verification statistics for different task execution time windows and exploration limits: the percentage denotes difference between WCET and BCET, limit is in terms of 250ms cycles (∞ stands for no limit, i.e. full exploration), memory in MB, time in hours:minutes:seconds.

limit	$f = 100\%$			$f = 95\%$			$f = 90\%$			$f = 86\%$		
cycle	states	mem	time	states	mem	time	states	mem	time	states	mem	time
1	0.001	51.2	1.47	0.5	83.0	15:03	1.5	124.1	1:22:43	3.3	186.9	6:39:47
2	0.003	53.7	2.45	0.8	96.8	27:00	2.4	139.7	2:09:15	5.3	198.7	9:14:59
4	0.005	54.5	4.62	1.5	97.2	48:02	4.4	138.3	3:48:40	9.2	274.6	14:12:57
8	0.010	54.7	8.48	2.8	97.8	1:28:45	9.1	156.5	8:38:42	18.2	364.6	28:35:32
16	0.020	55.3	16.11	5.4	112.0	2:45:52	17.8	176.0	16:42:05	35.4	520.4	44:06:57
∞	0.196	58.8	2:39.64	52.7	553.9	27:05:07	181.9	1682.2	147:23:25	**pos.unsafe**		99:07:56

Since symbolic MC proves that $f = 90\%$ case is safe, we also computed WCRT upper bounds. Table 4 compares the UPPAAL bounds on WCRTs with the bounds from classical response time analysis performed by Terma A/S. In particular Terma A/S found that `PrimaryF` task (#21) might violate its deadline even though this violation has never been observed neither in simulations nor in system deployment, whereas the bound provided by Uppaal is still within the deadline, thus (re)confirming schedulability.

Table 4: Specification and worst-case response-times of individual tasks.

		Specification			WCRT			
ID	Task	Period	WCET	Deadline	Terma	$f = 100\%$	$f = 95\%$	$f = 90\%$
1	RTEMS_RTC	10.000	0.013	1.000	0.050	0.013	0.013	0.013
2	AswSync_SyncPulseIsr	250.000	0.070	1.000	0.120	0.083	0.083	0.083
3	Hk_SamplerIsr	125.000	0.070	1.000	0.120	0.070	0.070	0.070
4	SwCyc_CycStartIsr	250.000	0.200	1.000	0.320	0.103	0.103	0.103
5	SwCyc_CycEndIsr	250.000	0.100	1.000	0.220	0.113	0.113	0.113
6	Rt1553_Isr	15.625	0.070	1.000	0.290	0.173	0.173	0.173
7	Bc1553_Isr	20.000	0.070	1.000	0.360	0.243	0.243	0.243
8	Spw_Isr	39.000	0.070	2.000	0.430	0.313	0.313	0.313
9	Obdh_Isr	250.000	0.070	2.000	0.500	0.383	0.383	0.383
10	RtSdb_P_1	15.625	0.150	15.625	4.330	0.533	0.533	0.533
11	RtSdb_P_2	125.000	0.400	15.625	4.870	0.933	0.933	0.933
12	RtSdb_P_3	250.000	0.170	15.625	5.110	1.103	1.103	1.103
13	(no task, this ID is reserved for priority ceiling)							
14	**FdirEvents**	250.000	5.000	230.220	7.180	5.553	5.553	5.553
15	**NominalEvents_1**	250.000	0.720	230.220	7.900	6.273	6.273	6.273
16	**MainCycle**	250.000	0.400	230.220	8.370	6.273	6.273	6.273
17	HkSampler_P_2	125.000	0.500	62.500	11.960	5.380	7.350	8.153
18	HkSampler_P_1	250.000	6.000	62.500	18.460	11.615	13.653	14.153
19	Acb_P	250.000	6.000	50.000	24.680	6.473	6.473	6.473
20	IoCyc_P	250.000	3.000	50.000	27.820	9.473	9.473	9.473
21	**PrimaryF**	250.000	34.050	**59.600**	**65.47**	54.115	56.382	58.586
22	**RCSControlF**	250.000	4.070	239.600	76.040	53.994	56.943	58.095
23	Obt_P	1000.000	1.100	100.000	74.720	2.503	2.513	2.523
24	Hk_P	250.000	2.750	250.000	6.800	4.953	4.963	4.973
25	StsMon_P	250.000	3.300	125.000	85.050	17.863	27.935	28.086
26	TmGen_P	250.000	4.860	250.000	77.650	9.813	9.823	9.833
27	Sgm_P	250.000	4.020	250.000	18.680	14.796	14.880	14.973
28	TcRouter_P	250.000	0.500	250.000	19.310	11.896	11.906	14.442
29	Cmd_P	250.000	14.000	250.000	114.920	94.346	99.607	101.563
30	**NominalEvents_2**	250.000	1.780	230.220	102.760	65.177	69.612	72.235
31	**SecondaryF_1**	250.000	20.960	189.600	141.550	110.666	114.921	122.140
32	**SecondaryF_2**	250.000	39.690	230.220	204.050	154.556	162.177	165.103
33	Bkgnd_P	250.000	0.200	250.000	154.090	15.046	139.712	147.160

4 Statistical Analysis

In the previous section, we observed that symbolic MC can be used to conclude schedulability, but not to disprove it. This is reflected in the first line of Table 1 where there is a wide range of values of f for which symbolic MC cannot conclude due to the potential presence of spurious counterexamples. In this section, we introduce SMC, a technique that we consider here to be the dual of symbolic MC. Namely, SMC can be used to disprove schedulability, but not to conclude it.

Concretelly, SMC is a simulation-based approach whose core objective is to estimate the probability for a system to satisfy a property by simulating and observing some of its executions, and then apply statistical algorithms to obtain the result. SMC is parameterized by two parameters: a *confidence interval size* on the estimate of the probability and a *confidence level* on the probability that the answer returns by methodology is correct. In terms of schedulability, SMC will thus be useful to generate concrete counterexample but cannot be used to conclude schedulability.

Several SMC algorithms have recently been implemented in UPPAAL [DLL+11b]. In this section, we will show how this implementation can be used not only to prove schedulability, but also to observe and reason on the execution of tasks. The latter will be done by exploiting the simulation engine and various informations displayed by the GUI of the tool.

SMC relies on the assumption that the dynamic of the system is entirely stochastic. In [DLL+11a,DLL+11b], we have proposed a refined stochastic semantic for timed automata that associates probability distributions to both the time-delays spend in a given state as well as to the transition between states. In this semantics timed automata components repeatedly race against each other, i.e. they independently and stochastically decide on their own how much to delay before outputting, with the "winner" being the component that chooses the minimum delay. Our stochastic schedulability model exploits the semantic of [DLL+11a,DLL+11b] as it assumes the execution time of the task to be picked uniformly in the interval $[f \cdot \mathrm{WCET}_i, \mathrm{WCET}_i]$.

In the rest of this section, we shall see how the SMC approach can be used to generate a witness traces when concluding that the system is not schedulable with a probability greater than 0. We will also illustrate how the SMC engine of UPPAAL can evaluate the probability to reach a state violating a deadline. Finally, and not to be underestimated, we will show how the GUI of the UPPAAL tool can be exploited to give quantitative feedback to the user on, e.g., blocking time, CPU usage, distribution of response time.

Running Example. Table 5 shows the query used to evaluate the probability of violating a deadline for runs bounded by 200 time units and the results for different values of f. We check only for cases when the symbolic model-checker reports that deadlines *may* be violated to generate a witness with SMC. The SMC technique gives results with certain levels of confidence and precision, i.e., the actual result is an interval. However, if the lower bound is strictly positive, it guarantees that the checker did find witnesses. The case $f = 80\%$ is interesting because it seems to be a spurious result from the symbolic model-checker. In fact we can do hypothesis testing to get a more precise result more cheaply. The model-checker accepts the hypothesis `Pr[<=200](<> error) <= 0.00001` with 1% significance level in 25s. As summarized in line 2 of Table 1 SMC allow to conclude unschedulability for $f \leq 79\%$.

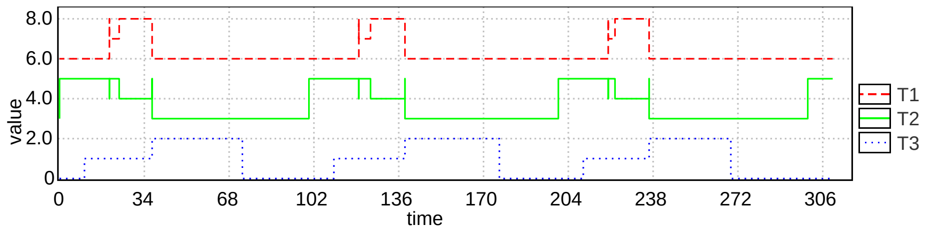
Table 5: Probability of error estimation with 1% level of significance.

f	50%	70%	79%	80%
`Pr[<=200](<> error)`	[0.847,0.858]	[0.604,0.615]	[0.301,0.312]	[0,0.005]

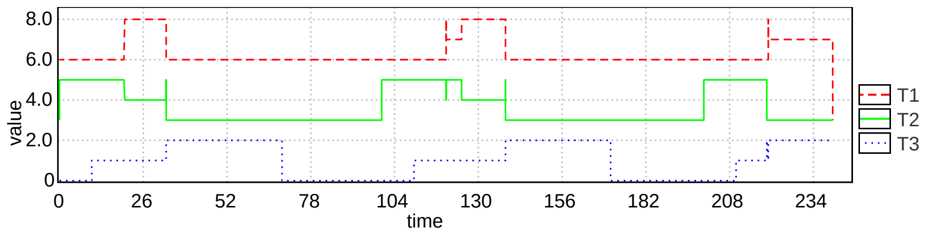
We can visualize traces (and inspect witnesses of deadline violation) by asking the checker to generate random simulation runs and visualize the value of a collection of expressions as a function time in a Gantt chart. In addition, we can filter these runs and only retain some that reach some state, here the error state. This is done with the following query producing the plot in Fig. 4b:

```
simulate 1000 [<=300] {
  (T(1).Ready+T(1).Computing+T(1).Release+runs[1]-2*T(1).Error)+6,
  (T(2).Ready+T(2).Computing+T(2).Release+runs[2]-2*T(1).Error)+3,
  (T(3).Ready+T(3).Computing+T(3).Release+runs[3]-2*T(1).Error)+0
} :1: error
```

If the filtering ("`:1:error`") is omitted, the plot contains all the runs, and for clarity just a single of them is displayed in Fig. 4a. As a result the plot encodes the



(a) Normal run using $f = 80$.



(b) Failed run using $f = 79$.

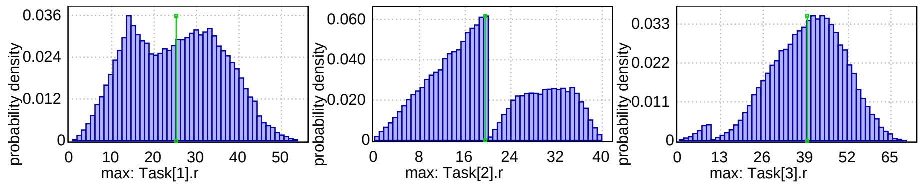
Fig. 4: Visualization of runs as a Gantt chart. The chart shows an encoding of the state with different weights corresponding to steps of different heights.

task states (idle, ready, running or error) in the level of the curve. For example, Figure 4a shows that T2 becomes ready and running starting from 0 time. At 10 task T3 becomes ready, but is not running. Then at 20 task T1 becomes ready and becomes running by preempting T2 but then it immediately gives up the running status (due to resource blocking) and resumes by preemption when T2 releases the resource. At this point T2 is not finished yet and will be able to finish only when T1 finishes and releases the CPU, hence there is a small spike just before going to idle state. The lowest priority task T3 has a chance to run and finish only when both T1 and T2 are done. Figure 4b is interpreted similarly, where the task T1 violates its deadline because T3 managed to get the resource before T1 and thus T1 was blocked from finishing.

More insight on the behavior of the tasks is gained by estimating expected response times using the queries:

```
E[<=200; 50000] (max: T(1).r)
E[<=200; 50000] (max: T(2).r)
E[<=200; 50000] (max: T(3).r)
```

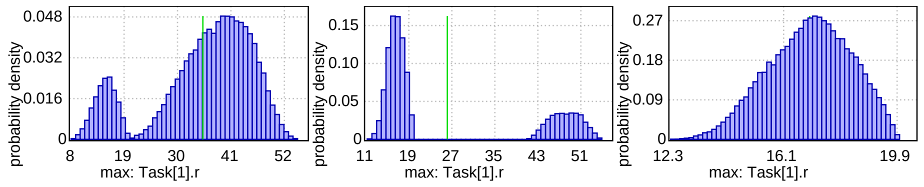
The result is the response time averages respectively: 16.96, 36.96 and 63.65 time units. In addition tool provides the probability densities shown in Figures 5. The plots show the effect of priority inversion on the higher priority tasks that may be delayed by the lower priority task.



(a) Task `T1`. (b) Task `T2`. (c) Task `T3`.

Fig. 5: Response time distributions for the different tasks when $f = 0$.

The response of `T1` goes beyond the deadline for $f = 0\%$, thus we evaluate the shapes of response time distributions for various f values in Fig. 6. Surprisingly there is a sharp contrast between $f = 79\%$ (unsafe for sure) and $f = 80\%$ which does not seem to exhibit the error and responds within 20 time units. This worst response time is more optimistic than the case $f = 83\%$ from symbolic analysis, which suggests that the symbolic analysis most probably is not exact for $f \in [80, 83]$. Figure 6a is an intermediate result between $f = 0\%$ (Fig. 5a) and $f = 79\%$ where the two seemingly normal "hills" are wide enough to meet each other, thus Fig. 5a is the result of two "hills": one from safe responses and the other slipped beyond a safety threshold but they are overlapping so tightly that this fact is hardly evident in Fig. 5a.



(a) $f = 50\%$ (not safe). (b) $f = 79\%$ (not safe). (c) $f = 80\%$ (seem OK).

Fig. 6: Response time distributions for Task `T1` using various f ratios.

Herschel. We generalize this methodology to our more complex Herschel case-study to confirm deadline violations and to study performance.

Table 6: Results of Herschel statistical model-checking.

Limit cycles	f %	SMC parameters α	ε	Total traces, #	Error traces #	Probability	Earliest cycle	Error offset	Verification time
1	0	0.0100	0.005	105967	1928	0.018194	0	79600.0	1:58:06
1	50	0.0100	0.005	105967	753	0.007106	0	79600.0	2:00:52
1	60	0.0100	0.005	105967	13	0.000123	0	79778.3	2:01:18
1	62	0.0005	0.002	1036757	34	0.000033	0	79616.4	19:52:22
160	63	0.0100	0.05	1060	177	0.166981	0	81531.6	2:47:03
160	64	0.0100	0.05	1060	118	0.111321	1	79803.0	2:55:13
160	65	0.0500	0.05	738	57	0.077236	3	79648.0	2:06:55
160	66	0.0100	0.05	1060	60	0.056604	2	82504.0	2:62:44
160	67	0.0100	0.05	1060	26	0.024528	1	79789.0	2:64:20
160	68	0.0100	0.05	1060	3	0.002830	67	81000.0	2:67:08
640	69	0.0100	0.05	1060	8	0.007547	114	80000.0	12:23:00
640	70	0.0100	0.05	1060	3	0.002830	6	88070.0	12:30:49
1280	71	0.0100	0.05	1060	2	0.001887	458	80000.0	25:19:35

Table 6 shows the results when we vary the execution time to be in the interval $[f \cdot \mathrm{WCET}, \mathrm{WCET}]$. The table shows the probabilities in function of this factor f and statistical parameters α ($1-\alpha$ is the confidence level) and ε the size of the confidence interval of the probability. Asking for more precise results yields more traces at the cost of time. At first we limited the search to just one cycle of 250ms, but then at the point of $f = 62\%$ the errors are rarely found even with high confidence and many runs. Then we increased the limit which increased our chances of finding the errors, we were lucky to find some errors as early as in the first cycle. Most of the errors are found quite early (cases where $f < 68$), but for smaller time-windows it is much harder to find and the few found ones are quite far in the run. Eventually the search took more than a day to find only a few error instances for $f = 71\%$, hence we stopped here.

Similarly to Fig. 5, response times for the most stressed task `PrimaryF` are estimated by generating 2000 probabilistic runs limited to 156 cycles for the safe case of $f = 90\%$. The vast majority (1787) of instances responded before 51093.3 and the rest is distributed about evenly (see Fig. 7). The worst found response time was of 52851.2 which is significantly lower than bound of 58586.0 found by symbolic MC in Table 4. The computation for this model took 17.6 hours.

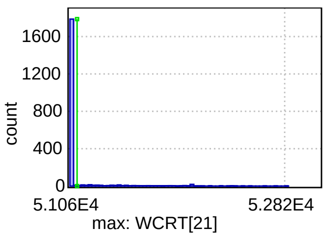
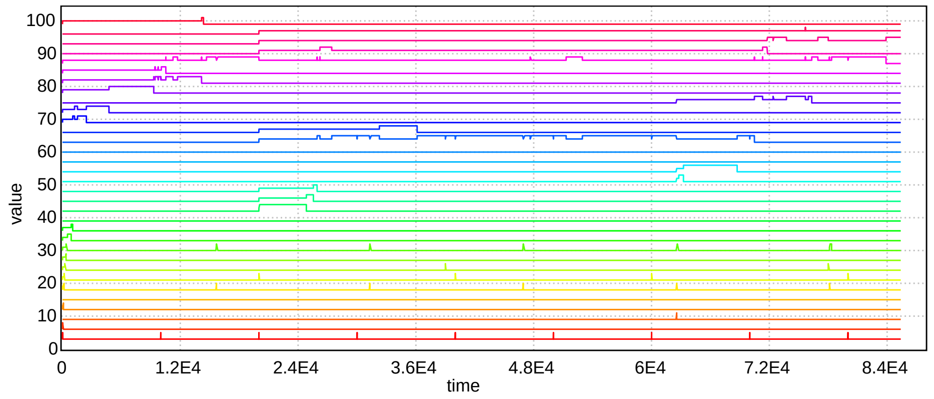


Fig. 7: Response times.

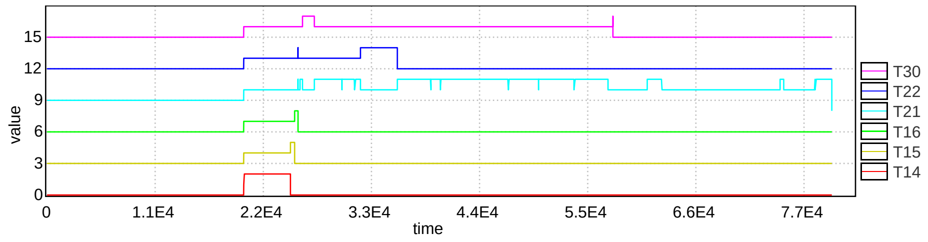
Fig. 8a shows an overview chart of all 32 tasks during the first 85ms. Each task can be identified by its base level 3*ID, thus `PrimaryF` with ID=21 is at 63. `PrimaryF` starts with an offset of 20ms and it has to finish before a deadline of 59.6ms. Under safe conditions of $f = 90\%$ `PrimaryF` finishes before $70500\mu s$ (Fig. 8a) but with $f = 50\%$ it fails at $79828.3\mu s$ (Fig. 8b).

5 Conclusion

In this paper, we have applied both symbolic MC and statistical MC to schedulability analysis. In particular, we have demonstrated that the complementary



(a) A successful run with $f = 90$ (`PrimaryF` at level 63).



(b) Selected processes of a simulation run with $f = 50\%$, where `PrimaryF` (task `T21` at level 9) violates a deadline.

Fig. 8: The first 85ms of Herschel model simulation runs.

qualities of the two methods allow to conclusively confirm as well as disprove schedulability for a wide range of cases. This is an impressive result as the problem is known to be undecidable. In addition we have illustrated how the user can benefit from the UPPAAL features in plotting, observing and reasoning about task executions, and hence improving the modeling process. We also believe that the combination of symbolic MC and statistical MC will prove highly useful in analyzing systems with mixed critically, i.e. systems containing tasks with hard timing constraints as well as soft, where the timing constraints are permitted to be violated occasionally.

References

BACC$^+$98. Hanene Ben-Abdallah, Jin-Young Choi, Duncan Clarke, Young Si Kim, Insup Lee, and Hong-Liang Xie. A process algebraic approach to the schedulability analysis of real-time systems. *Real-Time Systems*, 15:189–219, 1998. 10.1023/A:1008047130023.

BDL$^+$12. Peter E. Bulychev, Alexandre David, Kim Guldstrand Larsen, Axel Legay, Marius Mikučionis, and Danny Bøgsted Poulsen. Checking and distributing statistical model checking. In *NASA Formal Methods*, volume 7226 of *Lecture Notes in Computer Science*, pages 449–463. Springer, 2012.

BHK99. Steven Bradley, William Henderson, and David Kendall. Using timed automata for response time analysis of distributed real-time systems. In *Systems ," in 24th IFAC/IFIP Workshop on Real-Time Programming WRTP 99*, pages 143–148, 1999.

BHK$^+$04. H.C. Bohnenkamp, H. Hermanns, R. Klaren, A. Mader, and Y.S. Usenko. Synthesis and stochastic assessment of schedules for lacquer production. In *Quantitative Evaluation of Systems, 2004. QEST 2004. Proceedings. First International Conference on the*, pages 28 – 37, sept. 2004.

BHM09. Aske Brekling, Michael R. Hansen, and Jan Madsen. Moves – a framework for modelling and verifying embedded systems. In *Microelectronics (ICM), 2009 International Conference on*, pages 149–152, dec. 2009.

Bur94. Alan Burns. *Principles of Real-Time Systems*, chapter Preemptive priority based scheduling: An appropriate engineering approach, page 225–248. Prentice Hall, 1994.

DILS10. Alexandre David, Jacob Illum, Kim G. Larsen, and Arne Skou. Model-based design for embedded systems. In Gabriela Nicolescu and Pieter J. Mosterman, editors, *Model-Based Design for Embedded Systems*, chapter Model-Based Framework for Schedulability Analysis Using UPPAAL 4.1, pages 93–119. CRC Press, 2010.

DLL$^+$11a. Alexandre David, Kim G. Larsen, Axel Legay, Marius Mikučionis, Danny Bøgsted Poulsen, Jonas Van Vliet, and Zheng Wang. Statistical model checking for networks of priced timed automata. In *FORMATS*, LNCS, pages 80–96. Springer, 2011.

DLL$^+$11b. Alexandre David, Kim G. Larsen, Axel Legay, Zheng Wang, and Marius Mikučionis. Time for real statistical model-checking: Statistical model-checking for real-time systems. In *CAV*, LNCS. Springer, 2011.

FKPY07. Elena Fersman, Pavel Krcal, Paul Pettersson, and Wang Yi. Task automata: Schedulability, decidability and undecidability. *Information and Computation*, 205(8):1149 – 1172, 2007.

JP86. Mathai Joseph and Paritosh K. Pandya. Finding response times in a real-time system. *Comput. J.*, 29(5):390–395, 1986.

KZH$^+$09. J-Pieter Katoen, I. S. Zapreev, E. Moritz Hahn, H. Hermanns, and D. N. Jansen. The ins and outs of the probabilistic model checker MRMC. In *Proc. of 6th Int. Conference on the Quantitative Evaluation of Systems (QEST)*, pages 167–176. IEEE Computer Society, 2009.

LDB10. Axel Legay, Benoît Delahaye, and Saddek Bensalem. Statistical model checking: An overview. In *RV*, volume 6418 of *Lecture Notes in Computer Science*, pages 122–135. Springer, 2010.

MLR$^+$10. Marius Mikučionis, Kim Guldstrand Larsen, Jacob Illum Rasmussen, Brian Nielsen, Arne Skou, Steen Ulrik Palm, Jan Storbank Pedersen, and Poul Hougaard. Schedulability analysis using uppaal: Herschel-planck case study. In Tiziana Margaria, editor, *ISoLA 2010 – 4th International Symposium On Leveraging Applications of Formal Methods, Verification and Validation*, volume Lecture Notes in Computer Science. Springer, October 2010.

SLC06. Oleg Sokolsky, Insup Lee, and Duncan Clarke. Schedulability analysis of aadl models. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, page 8 pp., april 2006.

SVA04. Koushik Sen, Mahesh Viswanathan, and Gul Agha. Statistical model checking of black-box probabilistic systems. In *CAV*, LNCS 3114, pages 202–215. Springer, 2004.

YS06. Håkan L. S. Younes and Reid G. Simmons. Statistical probabilistic model checking with a focus on time-bounded properties. *Inf. Comput.*, 204(9):1368–1409, 2006.