Evaluating SMT Solvers on Schedulability Checking Instances

Ravindra Metta[1,2][0000−0001−7368−2389], Anand Yeolekar[1,2][0009−0002−3311−8809], and Samarjit Chakraborty[3][0000−0002−0503−6235]

[1] TCS Research, Tata Consultancy Services, Pune, India
[2] Technical University of Munich, Germany
r.metta@tcs.com, anand.yeolekar@tcs.com

[3] The University of North Carolina at Chapel Hill, USA
samarjit@cs.unc.edu

Abstract. When using an off-the-shelf Satisfiability Modulo Theory (SMT) solver for solving decision problems, we empirically demonstrate that solver performance is dependent on a number of factors. In particular, these are a combination of (i) constraints modeling the application behaviour, (ii) non-determinism inherent at the application level and the encoding of multiple system behaviours, and (iii) presence (absence) of witnesses, and these can affect solver performance in non-intuitive ways. Further, there is a wide variation in solver performance, *e.g.,* in terms of time and memory consumed across different solvers. Based on controlled experiments on analyzing schedulability problems encoded as SMT formulas, we compare the performance of selected state-of-the-art solvers. Our experiments help develop insights into understanding solver behaviour for domain-specific SMT instances, and help assess the impact of non-determinism on solver performance.

Keywords: Schedulability analysis · SMT solvers · Real-Time Systems

1 Introduction

Real-time systems form an important and large class of software systems. Ubiquitously present in embedded and cyber-physical systems, these systems span diverse application domains such as consumer electronics, automotive and avionics control software, medical devices, and telecommunications. A central requirement in the design, development and validation of real-time systems is that the system meets specified *timing properties*, in addition to satisfying functional properties. A common example of such a timing property is that the task set should be *schedulable* under *all* execution scenarios, *i.e.,* all task instances should meet their respective deadlines in any observation window. In particular, systems classified as *safety-critical* need to be rigorously checked for compliance to stringent timing properties. Thus, *exact* schedulability analysis is desirable for this class of systems.

An instance of a task is called a job. Jobs of a task are released at regular time intervals to be executed on a processor, and a *task specification* describes task parameters like the task offset, jitter, and job release patterns (*e.g.,* periodic or sporadic) along with their parameters such as a period or minimum inter-arrival time. Additional parameters include deadlines, that might also be implicitly or externally specified, and the best- and worst-case execution times of the jobs released by the task. The task set *deployment specification* describes the external environment such as the execution platform (that may consist of unicore or multicore processors), a scheduling policy, inter-task dependencies, and whether jobs can be preempted before completion and resumed again. A *run* of the tasks describes an execution of the set of tasks consisting of the release, start, end and deadline times of the task instances, along with auxiliary details like the processor-to-task mapping. Note that the task specification can admit *non-determinism* via release jitter, execution time variation, selection of arbitrary equal-priority ready-to-run task instances, or arrival of sporadic job instances of a talk. This leads to multiple runs of jobs in a task set, and a deadline miss could occur along any run.

A number of techniques have been developed for schedulability analysis of classes of real-time task sets. However, most of these are pessimistic or report approximate results in the presence of non-determinism. This is especially problematic in the case of modern cyber-physical systems such as autonomous cars [5, 7]. These systems have complex and distributed architectures with different communication protocols [28]. Further, control tasks implementing autonomous features have conditional control flows that make timing and schedulabuity analysis to be more challenging [6, 4]. Hence, scheduling [14, 29, 38] and timing verification [20] for such setups have drawn considerable attention in the past. Because of the complexity of tight timing and schedulability analysis, some studies have focused on relaxing the timing constrains required for guaranteeing the safety of feedback controllers [13] or have proposed techniques for reducing the timing interference across critical tasks [22]. Rigorous, formal methods-based approaches for schedulability analysis have also been proposed [34, 35, 33, 26, 16, 15, 17]. These approaches encode runs of the task set along with conditions for deadline miss, either as a model checking problem, or a set of constraints. The encoding is then submitted to off-the-shelf model checkers or SMT solvers such as Cbmc [19], Z3 [24], or Yices [11], to detect deadline misses.

One of the challenges in such formal modeling and analysis is to scale the approach to analyze large task sets. The work presented in [34, 35] is currently capable of analyzing small to medium-sized task sets. Formal methods based approaches, while being precise in their analysis and results, struggle with the well-known state space explosion problem. Hence, it is necessary to identify analysis bottlenecks and opportunities to improve their overall efficiency. With this as the motivation for this work, we focus on gaining insights into the performance of modern SMT solvers with respect to the SMT instances arising from a schedulability analysis encoding. In particular, we have studied encodings extended from [34] and [35]. These encodings supports task set execution over multicore platforms and admit sporadic task instances. Our objectives in this empirical study are to address the following research questions:

- **RQ1:** Understand how task complexity affects solver performance *e.g.,* number of tasks, and number of task instances.
- **RQ2:** Understand how varying task parameters that induce non-determinism in the system behaviour, affect solver performance
- **RQ3:** Understand how solver performance varies with respect to time and memory, and the presence (absence) of witnesses

Towards addressing the above research questions, we adopt the following experimentation methodology:

- Design candidate task sets which will serve as representative benchmarks of SMT instances for the purpose of schedulability analysis
- Conduct controlled experiments by varying task parameters, especially those that impact the system state-space
- Observe solver performance with default settings

2 SMT encoding for schedulability analysis

We briefly present the schedulability encoding, that has been extended from previous works [34, 35]. In these works, the schedulability analysis for detecting task deadline misses was restricted to task sets consisting of only periodic tasks, deployed on a unicore platform. However, real-world task sets consist of a mix of periodic and sporadic tasks, deployed on multicore execution platforms [31, 1]. Hence, the schedulability analysis tool is expected to support this class of task sets.

Let $\tau_i \in \mathcal{T}$ be a task specified as:

$$\tau_i \coloneqq \langle i, O, J, \mathit{type}, P, \underline{C}, \overline{C} \rangle \tag{1}$$

where i is a unique task id; O is the task offset indicating the delay in release of the first instance of this task; J is the maximum release jitter *i.e* each task instance may experience a delay in release by at most J time units from its scheduled release time; $\mathit{type} \in \{\mathit{per}, \mathit{spo}\}$ indicates if the task is periodic; P denotes the task period if $\mathit{type} = \mathit{per}$, or denotes the minimum inter-arrival time between consecutive sporadic instances; and $\underline{C}, \overline{C}$ denotes that best- and worst-case execution times of the task *i.e* each task instance may terminate at any time point in this interval.

Let H denote the analysis horizon, thus task instances are spawned (symbolically) up to time H. Each task instance is associated with symbolic variables $r_{i,j}, s_{i,j}, e_{i,j}, d_{i,j}, p_{i,j}$ that denotes respectively, the release time, start, end, deadline time, and processor allocation, of the j-th instance of task i. We assume all times are intervals are *discrete*, thus they can be encoded as integers with appropriate scaling of the unit. We assume $p_{i,j} \in \{1, 2, \ldots, N\}$ where N is the number of processors on which task instances could be deployed by the scheduler. Further, the processors are assumed to be identical and the scheduler is free to assign any available processor to the next ready-to-run task instance. In this work, we assume the scheduling policy is non-preemptive earliest-deadline-first (NP-EDF).

Definition 1. *A run of the task set is a sequence of task instances, sorted on start time* $\langle \ldots, (s_{i,j}, e_{i,j}, p_{i,j}), \ldots \rangle$*, respecting the scheduling policy and processor work conservation.*

The goal of the encoding and subsequent analysis is to check if any task instance misses deadline within the analysis horizon H. Figure 1 denotes the tool flow indicating processing steps involved in the analysis. The freshly introduced symbolic variables (`Symbolic vars` box in Fig. 1), associated with each task instance spawned up to the horizon H, are unconstrained. Appropriate constraints are introduced on these variables such that the solutions to the constraints correspond exactly to the set of *valid* runs of the task set $\mathcal{T}$ (`Encoding Runs` box in Fig. 1). Here, we only briefly list some of the constraints involved.

Constraints on release time A task instance can't be scheduled unless released: $r_{i,j} \leq s_{i,j}$. The release time of every task instance is affected by jitter: $O_i + j \times P_i \leq r_{i,j} \leq j_i + O_i + j \times P_i$.

Constraint on execution time For each task instance, the execution time is bounded between the best- and worst-case values as follows: $\underline{C}_i \leq e_{i,j} - s_{i,j} \leq \overline{C}_i$. Additionally, the processor must be within the specified range: $1 \leq p_{i,j} \leq N$.

Constraint on deadline time The deadline for periodic task instances is constrained as: $d_{i,j} = O_i + (j+1) \times P_i$. And for sporadic instances, it is specified as: $d_{i,j} = r_{i,j} + P_{i,j}$. In addition, for all sporadic task instances: $r_{i,0} \leq O_i$, $r_{i,j} \leq r_{i,j+1} + P_i$. Note that sporadic instances may or may not occur within the analysis horizon H.

Preventing overlap If two task instances, say (i,j) and (m,n), are scheduled on the same processor, they should not overlap in their execution. This is encoded as: $p_{i,j} = p_{m,n} \Rightarrow (e_{i,j} \leq s_{m,n} \vee e_{m,n} \leq s_{i,j})$

Global NP-EDF scheduling order If task instance (i,j) is scheduled earlier than instance (m,n) (irrespective of the processor assigned), then it must be that (i,j) had its deadline *no later than* that of (m,n), or (i,j) was scheduled before (m,n) was released *i.e* this pair of jobs did not contend with each other. This is encoded as: $s_{i,j} < s_{m,n} \Rightarrow (d_{i,j} \leq d_{m,n} \vee s_{i,j} < r_{m,n})$

Schedulability check The constraint that checks for deadline miss for any task instance is encoded as: $(e_{1,0} > d_{1,0} \vee \cdots \vee e_{n,k} > d_{n,k})$, assuming n tasks, with k being the last job spawned. Additionally, for sporadic instances, the deadline miss is valid only if the instance was released within the analysis horizon H.

The encoding step constructs these constraints for each task instance spawned within the horizon, or pairs of instances, as appropriate. These constraints are then composed via conjunction and submitted to the SMT solver. The full and detailed encoding with proof of correctness is beyond the scope of this work and will be presented elsewhere.

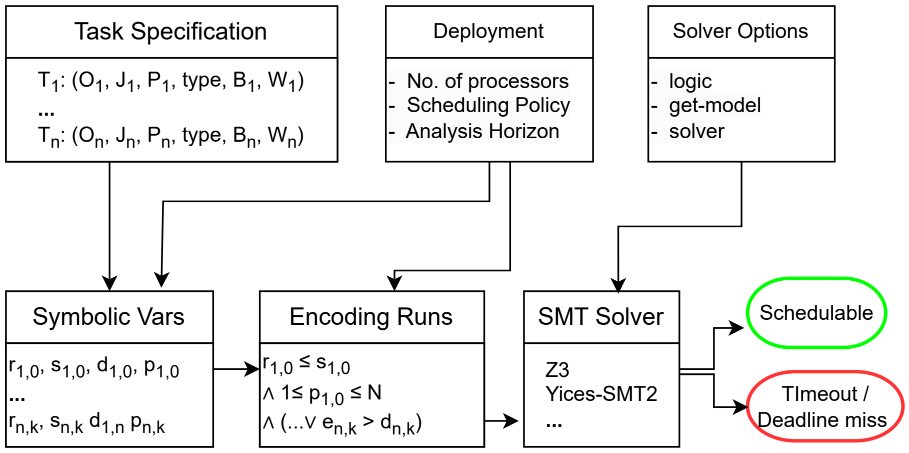

Fig. 1. Schedulability analysis tool flow

3 Experiments

The main focus of this work is to assess the impact of variation in task parameters, on the solver performance. In particular, application-level parameters that introduce non-determinism in the modeling and analysis, needs a systematic study. For this experimentation, we chose some of the top-performing SMT solvers from [32, 30]. In particular, we chose Z3 [25, 37], Yices-SMT2 [12, 36], CVC4 [3, 9], CVC5 [2, 10], Opemsmt [18, 27], MathSAT [8, 23]. These are also some of the most widely used SMT engines and are known to perform well in SMT competitions [30].

In order to assess the performance and suitability of the above SMT solvers, we conducted three sets of experiments to answer the research questions. We first took benchmark task sets from the Papabench, Malardalen, and Scade benchmarks presented in [21]. However, in [21] some of the task parameters such as jitter and best-case execution time are not specified. Therefore, we first created base versions of these task sets by carefully choosing the parameter values such that scheduling of the tasks according to NP-EDF policy will lead to at least one task instance missing its deadline. We term these task sets as *base* task sets. Their experimental evaluation is presented in Sec. 3.1.

Next, in order to assess the impact of task parameter-level non-determinism on SMT solvers, and gain insight into solvers are better suited for our encoding of schedulability analysis, we crafted variants of the base task sets by increasing (i) jitter, and (ii) worst-case execution time. The corresponding experimental evaluation is presented in Sec. 3.2.

Finally, any exhaustive schedulability check such as the SMT-based schedulability analysis technique briefly described in this paper, is expected to be exponentially difficult, as exact multicore schedulability analysis is known to be NP-hard. The overall state space corresponding to the possible scheduling sequences (*i.e* sequences of task instances, or task runs) increases exponentially with the input size *i.e* number of tasks or instances, in this case. The corresponding experimental evaluation is discussed in Sec. 3.3.

3.1 Base task sets

In order to conduct a realistic evaluation, we took three task sets: Papabench, a free real time benchmark developed for Unmanned Aerial Vehicles, Malardalen and Scade tasks are crafted by researchers for benchmarking analysis techniques. We created six task sets, each with at least one deadline violation, varying in size from 8 to 14 tasks, with processor choice varying from unicore to hexacore. Further, sporadic tasks are known to pose scalability challenges to schedulability analysis techniques. Therefore, we varied the number of sporadic tasks from 0 to 2. These task names can be seen in the column *Task* in Table. 1. The names of the task sets denote the particular composition of the task set. For example, `n1-t12-spo0.smt2` refers to the SMT encoding of a task set to be scheduled on a single core (`n1`), with 12 tasks (`t12`), out of which there are 0 sporadic tasks. In this table, *TO* represents *Time Out*.

We ran all the selected SMT solvers with default settings on these task sets with a 5 minute timeout. We limited the maximum number of total task instances within the analysis horizon to 100. The performance of the solvers in terms of time and memory is presented in Table. 1. Note that there is a deadline miss in each task set *i.e* each SMT instance has a witness or a satisfying assignment to the symbolic variables in the encoding. From this, we make the following observations:

- **Fastest solver** Z3 outperformed the other solvers and CVC5 is the slowest.
- **Increase in task complexity** Our set of experiments do not reveal any correlation or pattern with increase in number of processors or sporadic tasks, in the particular case of satisfiable instances. In general, solvers heuristics are tuned towards quickly discovering witnesses, and thus increase in task complexity does not seem to penalize the solver performance in the presence of witnesses.
- **Memory consumption** Yices-SMT2 is the least memory consuming among all the solvers
- **Overall** OpenSMT offers best trade off between time and memory

- **Wide variation** Performance varies widely across solvers for the same SMT instance, both in terms of memory and time
- **Hardness of the instances** In some instances, shallow witnesses were found quickly, even though the instance size or complexity may be "larger" than others *e.g.,* `n1`-t14-spo0 has a shorter witness as compared to `n1`-t12-spo0.

Task	**Z3**		**Y-SMT2**		**CVC4**		**CVC5**		**OpenSMT**		**MathSAT**	
	Tm	Mem	Tm	Mem	Tm	Mem	Tm	Mem	Tm	Mem	Tm	Mem
n1-t12-spo0	26	133	TO	29	61	137	TO	208	264	110	TO	111
n1-t14-spo0	0	65	0	9	1	42	5	64	1	19	0	52
n3-t8-spo2	10	169	34	38	53	170	107	223	34	192	73	182
n3-t14-spo2	3	112	36	23	50	137	153	178	60	61	38	116
n6-t12-spo2	6	162	23	27	16	151	76	189	17	65	23	163
n6-t14-spo2	70	107	124	21	70	106	207	148	98	66	114	109

Table 1. Solver Performance on base tasksets (Tm in *sec* and Mem in *MB)*

3.2 Solver performance with increasing non-determinism

In order to assess the impact of non-determinism introduced by varying task parameters on the solver performance, we varied the base task sets by increasing jitter and by increasing WCET. Jitter is the potential delay experienced by each task instance during its release. Thus, each task instance is released at a non-deterministic time point within the bounded range, instead of being release at a fixed periodic time instant. Further, each task instance can terminate non-deterministically at any time point in the interval [BCET,WCET]. Thus change in jitter and WCET parameters of the task impacts each corresponding task instance and thereby the set of runs of the task set. In our experimentation, we increased jitter and WCET of some of the randomly chosen tasks by at most 10%. The results are shown in Table. 2 and Table 3. Here, *B Tm* represents time taken for the base taskset, *J Tm* represents time taken for an increased jitter variation of the base taskset, *W Tm* represents time taken for an increased WCET variation of the base taskset, and *TO* represents timeout, which is restricted to 300 seconds. From this, we make the following observations:

- **Fastest solver** Z3 outperformed the other solvers and CVC5 is the slowest.
- **Increase in non-determinism due to jitter** In most cases, when the instances are satisfiable, increasing jitter is improving solver performance. This is due to the fact that, although the overall state space has increased due to jitter, but the solution space too has widened, as there are potentially more witnesses than earlier, compared to the base case. Hence, with more witnesses present in the state space, the solvers are generally able to discover a witness faster than the base case.
- **Increase in non-determinism due to WCET** Similar to the jitter parameter observation above, increasing WCET has a positive impact in this case. The reason for this is the same as explained above.

Task	Z3		Yices-SMT2		CVC4		CVC5		OpenSMT		MathSAT	
	B Tm	J Tm	B Tm	J Tm	B Tm	J Tm	B Tm	J Tm	B Tm	J Tm	B Tm	J Tm
n1-t12-spo0	27	4	TO	TO	62	147	TO	TO	265	224	TO	117
n1-t14-spo0	1	1	0	0	2	1	6	6	1	1	1	1
n3-t8-spo2	10	12	34	33	54	47	108	128	35	46	74	60
n3-t14-spo2	3	4	37	34	51	72	153	180	60	81	39	45
n6-t12-spo2	7	22	24	19	16	16	76	100	18	22	24	33
n6-t14-spo2	70	16	125	66	111	57	207	131	98	166	155	TO

Table 2. Solver Performance with increasing jitter (Tm is time in sec)

Task	Z3		Yices-SMT2		CVC4		CVC5		OpenSMT		MathSAT	
	B Tm	W Tm	B Tm	W Tm	B Tm	W Tm	B Tm	W Tm	B Tm	W Tm	B Tm	W Tm
n1-t12-spo0	27	51	TO	208	62	TO	TO	TO	265	139	TO	TO
n1-t14-spo0	1	1	0	0	2	2	6	5	1	1	1	2
n3-t8-spo2	10	13	34	33	54	59	108	164	35	36	74	47
n3-t14-spo2	3	6	37	38	51	49	153	154	60	80	39	41
n6-t12-spo2	7	10	24	13	17	17	76	120	18	23	24	72
n6-t14-spo2	70	6	125	31	111	47	207	105	98	63	155	82

Table 3. Solver Performance with increasing WCET (Tm is time in sec)

3.3 Solver performance for full state space exploration

Finally, we assess the impact of non-determinism on solver performance, when the task set is schedulable *i.e* no deadline miss. In this case, the solvers are forced to explore the entire state space *i.e* all the possible runs of the task set, stressing out the internal heuristics of the solvers. We created such task sets by carefully adjusting the values of task parameters such that the task set is schedulable. First, we created a set of tasks from a base task, by varying only the jitter parameter of different tasks. Then, we created another set of tasks by varying only the WCET of different tasks. Table 4 shows the results for a representative base task set `n4-t6-spo0`. Bother jitter and WCET increase was restricted to 10% of the initial values. Here, *TO* represents *Time Out*. From this, we make the following observations:

- **Fastest solver** Yices-SMT2, closely followed by OpenSMT, significantly outperformed the other solvers. Surprisingly, Z3, which performed the fastest given the presence of witnesses (deadline violations), is the slowest to explore the entire states space *i.e* unsatisfiable instances.
- **Memory-efficient** Yices-SMT2 is the least memory consuming solver across the instances, while the CVC twins require most memory.
- **Increase in non-determinism due to jitter** In all cases, when the instances are unsatisfiable, successively increasing jitter is negatively impacting solver performance, as expected. The solver has to explore increasingly larger state space to conclude schedulability.

- **Increase in non-determinism due to WCET** Here, we have a mix of results. In the case of CVC4 and CVC5, the observation matches that of jitter described above. However, in the case of Yices-SMT2, OpenSMT and MathSAT, with increasing number of tasks having a modified WCET, the solver performance is improving instead of degrading. This warrants more experimentation.

Task	**Z3**		**Y-SMT2**		**CVC4**		**CVC5**		**OpenSMT**		**MathSAT**	
	Tm	Mem	Tm	Mem	Tm	Mem	Tm	Mem	Tm	Mem	Tm	Mem
n4-t6-spo0-jitter	227.25	146	31.09	26	63.85	159	131.59	209	27.35	101	52.79	146
n4-t6-spo0-jitter-4t	TO	151	30.25	25	99.55	206	218.06	180	45.17	127	76.97	155
n4-t6-spo0	202.40	147	24.19	24	66.05	182	149.97	162	35.40	100	59.12	156
n4-t6-spo0-wcet1	TO	150	42.2	28	85.2	173	112.4	211	48.1	128	106.2	167
n4-t6-spo0-wcet2	TO	149	22.3	23	112.7	182	173.1	0	39.0	98	80.2	153

Table 4. Solver Perf. on full state-space exploration (Tm in *sec* and Mem in *MB)*

4 Concluding remarks

The availability of powerful SMT solvers have opened up the possibility of tight schedulability analysis for a variety of task models and complex architectures. But our study in this paper clearly shows that the performance of SMT solvers can vary significantly depending on the task complexity (RQ1), amount of non-determinism (RQ2), and full state space exploration (RQ3). Overall, Z3, Yices-SMT2 and OpenSMT seem to be better suited for our encoding of the schedulability analysis problems we considered. In the future, we intend to conduct more experiments on a wider range of task sets, with additional solvers.

References

1. Akesson, B., et al.: A comprehensive survey of industry practice in real-time systems. Real Time Systems **58**(3) (2022)
2. Barbosa, H., et al.: cvc5: A versatile and industrial-strength SMT solver. In: 28th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS) (2022)
3. Barrett, C.W., et al.: CVC4. In: 23rd International Conference on Computer Aided Verification (CAV) (2011)
4. Chakraborty, S., Erlebach, T., Thiele, L.: On the complexity of scheduling conditional real-time code. In: 7th International Workshop on Algorithms and Data Structures (WADS) (2001)
5. Chakraborty, S., Faruque, M.A.A., Chang, W., Goswami, D., Wolf, M., Zhu, Q.: Automotive cyber-physical systems: A tutorial introduction. IEEE Design & Test **33**(4), 92–108 (2016)
6. Chakraborty, S., Thiele, L.: A new task model for streaming applications and its schedulability analysis. In: Design, Automation and Test in Europe Conference ((DATE) (2005)
7. Chang, W., Chakraborty, S.: Resource-aware automotive control systems design: A cyber-physical systems approach. Foundations and Trends in Electronic Design Automation **10**(4), 249–369 (2016)

8. Cimatti, A., et al.: The MathSAT5 SMT Solver. In: 19th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS) (2013)
9. CVC4 Solver (2023), `https://cvc4.github.io/`
10. CVC5-website (2023), `https://cvc5.github.io/`
11. Dutertre, B.: Yices 2.2. In: 26th International Confernece on Computer-Aided Verification (CAV) (2014)
12. Dutertre, B.: Yices 2.2. In: 26th International Conference on Computer-Aided Verification (CAV) (2014)
13. Goswami, D., Schneider, R., Chakraborty, S.: Relaxing signal delay constraints in distributed embedded controllers. IEEE Trans. Contr. Sys. Techn. **22**(6), 2337–2345 (2014)
14. Goswami, D., et al.: Time-triggered implementations of mixed-criticality automotive software. In: Design, Automation & Test in Europe Conference & Exhibition (DATE) (2012)
15. Gu, Z.G., et al.: A model-checking approach to schedulability analysis of global multiprocessor scheduling with fixed offsets. International Journal of Embedded Systems (IJES) **6**(2-3) (2014)
16. Guan, N., et al.: Exact schedulability analysis for static-priority global multiprocessor scheduling using model-checking. In: Proceedings of 5th International Workshop on Software Technologies for Embedded and Ubiquitous Systems (SEUS) (2007)
17. Guan, N., et al.: Schedulability analysis of global fixed-priority or edf multiprocessor scheduling with symbolic model-checking. In: Proceedings of 11th International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC) (2008)
18. Hyvärinen, A.E.J., et al.: Opensmt2: An smt solver for multi-core and cloud computing. In: The International Conference on Theory and Applications of Satisfiability Testing (SAT) (2016)
19. Kroening, D., Tautschnig, M.: Cbmc – c bounded model checker. In: 20th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS) (2014)
20. Kumar, P., et al.: A hybrid approach to cyber-physical systems verification. In: Design Automation Conference (DAC) (2012)
21. Lunniss, W., Altmeyer, S., Davis, R.: A comparison between fixed priority and edf scheduling accounting for cache related pre-emption delays. Leibniz Transactions on Embedded Systems **1**(1) (2014)
22. Masrur, A., et al.: VM-based real-time services for automotive control applications. In: 16th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA) (2010)
23. MathSAT Solver (2023), `https://mathsat.fbk.eu/`
24. de Moura, L., Bjørner, N.: Z3: An efficient smt solver. In: 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS) (2008)
25. de Moura, L., Bjørner, N.: Z3: An efficient smt solver. In: 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS) (2008)
26. Nasri, M., Brandenburg, B.B.: An exact and sustainable analysis of non-preemptive scheduling. In: IEEE Real-Time Systems Symposium (RTSS) (2017)
27. OpenSMT Solver (2023), `https://verify.inf.usi.ch/opensmt`

28. Roy, D., et al.: Multi-objective co-optimization of FlexRay-based distributed control systems. In: 22nd IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS) (2016)
29. Schneider, R., et al.: Multi-layered scheduling of mixed-criticality cyber-physical systems. Journal of Systems Architecture - Embedded Systems Design **59**(10-D), 1215–1230 (2013)
30. SMTComp-2023 (2023), `https://smt-comp.github.io/2023/participants.html`
31. Sun, Z., Guo, M., Liu, X.: A survey of real-time scheduling on multiprocessor systems. In: 39th National Conference of Theoretical Computer Science, (NCTCS) (2021)
32. Weber, T., et al.: The SMT competition 2015-2018. Journal on Satisfiability, Boolean Modeling and Computation **11**(1) (2019)
33. Yalcinkaya, B., Nasri, M., Brandenburg, B.B.: An exact schedulability test for non-preemptive self-suspending real-time tasks. In: 23rd Design, Automation & Test in Europe Conference & Exhibition, (DATE) (2019)
34. Yeolekar, A., et al.: Refining task specifications using model checking. In: 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA) (2018)
35. Yeolekar, A., et al.: Checking scheduling-induced violations of control safety properties. In: 20th International Symposium on Automated Technology for Verification and Analysis (ATVA) (2022)
36. Yices-SMT2 Solver (2023), `https://yices.csl.sri.com/`
37. Z3 Solver (2023), `https://github.com/z3prover/z3.git`
38. Zhang, L., et al.: Task- and network-level schedule co-synthesis of ethernet-based time-triggered systems. In: 19th Asia and South Pacific Design Automation Conference (ASP-DAC) (2014)