Towards Reliable Benchmarks of Timed Automata

Rebeka Farkas, Gábor Bergmann
Budapest University of Technology and Economics, Department of Measurement and Information Systems
Email: {farkasr,bergmann}@mit.bme.hu
MTA-BME Lendület Cyber-Physical Systems Research Group

***Abstract*—The verification of the time-dependent behavior of safety-critical systems is important, as design problems often arise from complex timing conditions. One of the most common formalisms for modeling timed systems is the timed automaton, which introduces clock variables to represent the elapse of time. Various tools and algorithms have been developed for the verification of timed automata. However, it is hard to decide which one to use for a given problem as no exhaustive benchmark of their effectiveness and efficiency can be found in the literature. Moreover, there does not exist a public set of models that can be used as an appropriate benchmark suite. In our work we have collected publicly available timed automaton models and industrial case studies and we used them to compare the efficiency of the algorithms implemented in the Theta model checker. In this paper, we present our preliminary benchmark suite, and demonstrate the results of the performed measurements.**

I. Introduction

Since their introduction by Alur and Dill [1], timed automata has become one of the most common formalisms for modeling and verification of real time systems. There is a wide range of application areas, such as communication protocols [2] and digital circuits [3]. There are many extensions of the formalism, such as the *probabilistic timed automaton* that is able to represent stochastic behavior or the *parametric timed automaton* that can describe parametric timing properties.

The key challenge of the verification of timed automaton-based models is the same as in case of any formalism: developing efficient and scalable algorithms that can be applied in practice. Several algorithms and tools have been designed for this purpose, that may differ in the supported formalisms and queries. Since these algorithms are as diverse as the problems they address, a single best one can not be chosen: for each algorithm there are classes of models, that can be verified efficiently and other classes where other approaches would be more suitable to use. This raises the need for some guidelines to decide which tool to use for a given model.

A possible solution is to perform an exhaustive benchmark on a set of relevant problems that can later be used to determine which approach is the most suitable for a given problem, however, this would require a set of relevant case studies to use as inputs. Unfortunately such a benchmark suite is not available for timed automata.

Our goal is to provide a set of timed automaton models (and corresponding queries) that can be used as a benchmark suite for comparing the efficiency of tools and algorithms developed for the verification of timed automata. It is important that the benchmark suite should allow the performed measurements to fulfill the requirements of a reliable benchmark [4], [5].

In this paper we present our preliminary benchmark suite that we assembled based on this principle and we demonstrate its usability by measurements that we performed on the algorithms implemented in the Theta [6] model checker.

II. Background

A. Timed automata

1) Basic definitions: Clock variables are a special type of variables, whose value is constantly increasing as the time elapses. The only operation on clock variables is the *reset* operation that sets the value of a clock to a constant. It is an instantaneous operation, after which the value of the clock will continue to increase. The set of clock variables is denoted by $\mathcal{C}$.

A *clock constraint* is a conjunctive formula of *atomic clock constraints*. There are two types of atomic clock constraints:

- the *simple constraint* of the form $x \sim n$ and
- the *diagonal constraint* of the form $x - y \sim n$,

where $x, y \in \mathcal{C}$, $\sim \in \{\leq, <, =, >, \geq\}$ and $n \in \mathbb{N}$. In other words a clock constraint defines upper and lower bounds on the values of clocks and the differences of clocks. The set of clock constraints is denoted by $\mathcal{B}(\mathcal{C})$.

A *timed automaton* extends a finite automaton with clock variables. It is formally defined as a tuple $\langle L, l_0, E, I \rangle$ where

- L is the set of locations (i.e. control states),
- $l_0 \in L$ is the initial location,
- $E \subseteq L \times \mathcal{B}(\mathcal{C}) \times 2^{\mathcal{C}} \times L$ is the set of edges and
- $I : L \to \mathcal{B}(\mathcal{C})$ assigns invariants to locations [7].

The edges in E are defined by the source location, the guard (represented by a clock constraint), the set of clocks to reset, and the target location.

2) Extensions of timed automata: Many extensions have been invented to increase the descriptive and the expressive power of timed automata.

A *network of timed automata* [8] is the parallel composition of a set of timed automata. Communication is possible by shared variables or hand-shake synchronization using *actions* on the edges. Constructing networks of timed automata does not increase the expressive power, as a network of timed automata can be transformed into an equivalent timed automaton, but it does increase the understandability of the model.

A *parametric timed automaton* [9] is an extension of a timed automaton, where instead of constants, unbound parameters are also allowed to appear in clock constraints. Verification of

parametric timed automata focuses on finding the parameter bindings that satisfy certain properties.

A timed automaton extended with *data variables* (extended timed automaton [8]) is an extension, where besides clock variables, data variables (discrete variables such as integers, bools, etc.) are also allowed. Data variables can also appear in constraints to enable transitions (*data guards*) and can be modified by transitions (*update*). However, clock variables are not allowed to appear in data guards or updates.

3) Verification: The verification approach depends on the type of property to check. Analysing safety properties is reduced to searching for reachable states in the state space, while checking liveness properties is solved by looking for certain cycles (strongly connected components) in the state space. [10]

B. Reliable benchmarks

Many requirements of reliable benchmarks are described in the literature [4], [5]. Although most methodologies focus on the execution of the benchmarks and not the inputs, some of the requirements do raise expectations for the benchmark suite.

Realistic The inputs should resemble models from industrial case studies.

Simple Just like other aspects of the benchmark, the input models must be understandable.

Scalable Scalable models are necessary in order to support a wide range of approaches.

Portable For portability, the models and the properties should be defined in a widely supported format.

Public A reproducible benchmark requires a public benchmark suite.

Diverse In order to be able to analyze the strengths of a wide range of tools, the models and the problems should be classified and many classes of inputs should be included.

III. Related work

A. Model checking competitions

In case of most common formalisms, generally accepted benchmarks are carried out by model checking competitions, such as the Model Checking Contest[1] for Petri Nets, the SV-COMP[2] on software verification and the Hardware Model Checking Competition[3] for hardware models. These competitions are an effective way of assembling and maintaining realistic benchmark suites and performing reliable benchmarks. However, there is no such competition for timed automata.

B. Benchmarks of timed automata-based models

Since there does not exist a generally accepted benchmark suite, each tool uses its own set of inputs to demonstrate the efficiency of its algorithms. Table I summarizes the characteristics of input sets of the most common tools: name of the tool, input format, total number of models, number of scalable models, whether they describe a property to check, and whether they provide references to papers where these models are described. While most presented tools operate on networks of extended timed automata, CosyVerif's BenchKit [11] consists of parametric timed automata, and the PRISM benchmark suite contains probabilistic timed automata.

Tool	input	#models	#scalable	query	ref
UPPAAL	xml, xta	9	3	true	true
Kronos	aut	5	3	true	true
PAT	xml, xta	5	5	false	false
MCTA	xta	5	5	true	true
TChecker	xta	6	5	false	true
REDLIB	d	5	5	true	false
Shrinktech	aut	9	3	false	true
CosyVerif	grml	15	0	false	true
PRISM	pta	7	2	true	false

TABLE I
AVAILABLE BENCHMARK SUITES

As it can be seen in the table, most model checkers have their own input language. However, the most common input format is `xta` defined by UPPAAL [8]. The presented benchmark suites are small, and share many models – e.g. the scalable models are the same for all benchmark suites, except for MCTA that uses another kind of scalability (see Section IV-D). In many cases the properties to verify are not defined, instead, during benchmarks the complete statespace of the model is explored. In some cases the source and the description of the models are also missing.

In conclusion, the current benchmark suites are small, there are very few scalable models, and portability, diversity and understandability is not always ensured.

IV. Xta Benchmark Suite

In this section, we present the Xta Benchmark Suite that is a collection of inputs we propose for comparing the efficiency of timed automaton verification algorithms. The suite was constructed to meet the requirements enlisted in Section II-B. While Table II describes the contained models, the complete suite, including the models, the queries and the references can be found online[4]. Note, that this is a preliminary suite.

A. Sources

The benchmark suite consists of models from existing benchmarks of UPPAAL, PAT, MCTA and CosyVerif, as well as UPPAAL case studies and other public models. Industrial case studies were included in order to allow realistic benchmarks. References to papers describing the models are provided in order to ease understandability and to assure the benchmark inputs are realistic, public and portable.

B. Format

In order to help portability, the `xta` format was chosen for storing the models, as most timed model checkers are able to parse this format (even if they are not able to transform their own input language to `xta`). Another advantage of this format

[1] https://mcc.lip6.fr/

[2] https://sv-comp.sosy-lab.org/2018/

[3] http://fmv.jku.at/hwmcc17/

[4] https://github.com/farkasrebus/XtaBenchmarkSuite

Name	Description	Source	Type	Scalable
FISCHER	Fischer's mutual exclusion protocol	UPPAAL benchmark	MutEx protocol	true
CSMA	The CSMA/CD protocol	UPPAAL benchmark	CD protocol	true
FDDI	Token Ring/FDDI protocol	UPPAAL benchmark	protocol	true
BANDO	Bang-Olufsen protocol	UPPAAL benchmark	CD protocol	false
BOCDPFIXED	Bang-Olufsen Collision Detection Protocol	UPPAAL benchmark	CD protocol	false
BOCDP	BOCDP - original, faulty version	UPPAAL benchmark	CD protocol	false
CRITICAL	Critical region	PAT benchmark	MutEx protocol	true
LYNCH	Lynch-Shavit protocol	PAT benchmark	MutEx protocol	true
BAWCC	Business Agreement with Coordination Completion protocol	UPPAAL case studies	protocol	false
BAWCCENHANCED	BAWCC - enhanced version	UPPAAL case studies	protocol	false
SCHEDULE	Schedulability Framework model	UPPAAL case studies	algorithm	false
STLS	Single Tracked Line Segment	MCTA benchmark	system	false
MUTEX	Mutual exclusion protocol	MCTA benchmark	MutEx protocol	false
FAS	Fire Alarm System	[12]	system	false
SOLDIERS	The soldiers problem	public model	problem	false
ENGINE	A running engine	public model	system	false
ANDOR	And-Or circuit	CosyVerif BenchKit	circuit	false
BANGOLUFSEN	Bang-Olufsen protocol	CosyVerif BenchKit	protocol	false
EXSITH	Sluice	CosyVerif BenchKit	system	false
FLIPFLOP	Flip-flop circuit	CosyVerif BenchKit	circuit	false
LATCH	Latch circuit	CosyVerif BenchKit	circuit	false
MALER	Maler's Jobshop algorithm	CosyVerif BenchKit	algorithm	false
RCP	Root Connection Protocol	CosyVerif BenchKit	protocol	false
SIMOP	SIMOP Networked Automation System	CosyVerif BenchKit	system	false
SRLATCH	SR-latch circuit	CosyVerif BenchKit	circuit	false
TRAIN	Train gate controller protocol	CosyVerif BenchKit	system	false

TABLE II
XTA BENCHMARK SUITE MODELS

is that it is possible to define scalable models in a way that the size of the model can be modified by setting a single constant.

C. Transformations

In many cases the models were transformed. In case of timed automata, UPPAAL was used to transform the `xml`-based models to `xta`. Parametric timed automata are stored in an `xml` based format (`grml`) by CosyVerif, that was transformed to `xta` programmatically. As the `xta` format does not allow parameters, they were parsed as `const int` and manually bound to a value taken from the Shrinktech model of the same system, where it was available. Additionally, one of the models (*TRAIN*) was modified to a generalized version (that allows more trains) in order to increase scalability.

D. Scalability

In most existing benchmark suites, scalable models represent communication protocols and scalability is introduced by changing the number of participants – i.e. introducing new timed automata to the network that behave similarly to the original ones. On the other hand, in the MCTA benchmark suite scalability is introduced to the model by increasing constants used in clock constraints.

In the Xta Benchmark Suite only the former type of models are considered scalable, since the most commonly used algorithms are not sensitive to the values of bounds [7].

E. Queries

While algorithms based on state space exploration can operate without a property to check, in order for benchmarks to be realistic, the Xta Benchmark Suite also provides queries for most models. This allows to perform measurements on a wider range of algorithms – such as backward exploration, that requires the target state to initiate from, or search algorithms with heuristics that are efficient in finding an execution trace to the target state but inefficient in state space exploration.

F. Classification

In order to demonstrate diversity the models were classified according to the type of problem they represent. This also determines the types of properties to be checked. More information on the classification can be found online.

V. MEASUREMENTS

Algorithms implemented in the Theta model checking framework were executed on the benchmark suite. Unlike the usual purpose of benchmarks, these measurements were performed to analyze the benchmark suite and not the algorithms.

A. Procedure

The measurements were executed on a virtual 64 bit Windows 7 operating system with a 2 core CPU (2.50 GHz) and 4 GB of memory. Each algorithm was run 5 times on each input and the average of the runtimes was taken. The timeout was 5 minutes (300,000 milliseconds).

While the complete suite consists of 26 models, those without a property to check were excluded as well as the ones containing elements that Theta does not support yet, such as broadcast channels. This reduced the number of inputs to 11.

The efficiency of six algorithms were compared. Algorithm *LU* is presented in [13], algorithm *ACT* is the improvement of the algorithm described in [7] by applying the *activity* abstraction described in [14] using lazy evaluation and algorithms *BINITP, SEQITP, WEAKBINITP,* and *WEAKSEQITP* are variants of the algorithms described in [15].

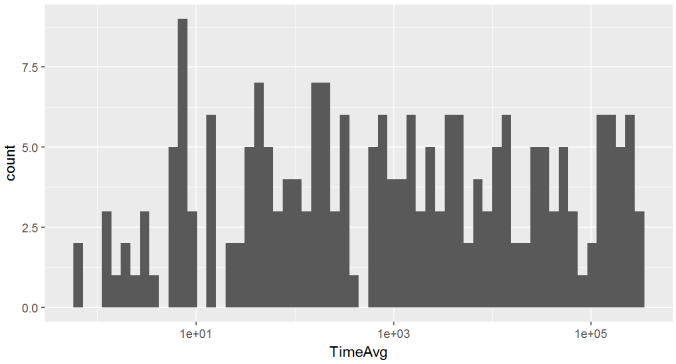

Fig. 1. Distribution of execution times

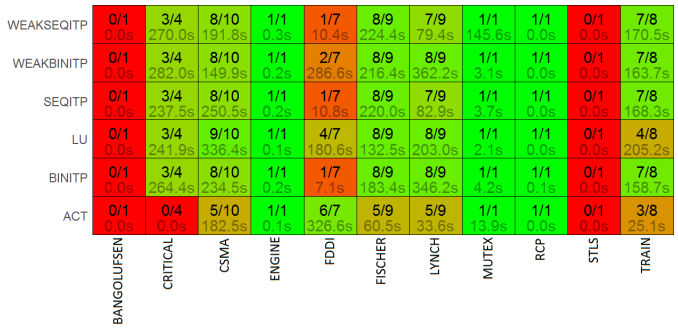

Fig. 2. Summary of results

B. Results

The distribution of execution times can be seen in Figure 1. Figure 2 summarizes the success rates and runtimes of the executions. The rows correspond to algorithms and the columns correspond to inputs.

The first row of a cell contains a fraction representing the success rate of the algorithm on the input: the denominator is the total number of instances of the model (in case of non-scalable models it is one) and the nominator is the number of instances successfully verified by the algorithm. The second row presents the runtime on the largest instance that was successfully verified or $0.0s$ if there was none.

Models *BANGOLUFSEN* and *STLS* turned out to be too large to be verified by any of the examined algorithms in the given time. Executing the algorithm *ACT* on input *CRITICAL* resulted in an exception for all instances.

Results show that for each algorithm the benchmark suite contained at least one model where the applicability of the algorithm was demonstrated and that the execution times are well distributed on the logarithmic scale.

VI. Conclusions

This paper proposed a benchmark suite to perform reliable benchmarks of verification algorithms for timed automata. The requirements of such a benchmark suite were identified, then a preliminary collection of inputs were presented. To demonstrate the applicability of the proposed benchmark suite the models were used to compare the algorithms implemented in the Theta model checking framework. The results of the presented benchmark suggest that the benchmark suite meets the described requirements.

Future works include increasing the size of the benchmark suite by importing more models from benchmarks of other tools or even other timed formalisms, focusing on scalable models. We also plan to include more industrial case studies that can be found in the literature.

References

[1] R. Alur and D. L. Dill, "The theory of timed automata," in *Real-Time: Theory in Practice, REX Workshop, Mook, The Netherlands, June 3-7, 1991, Proceedings*, 1991, pp. 45–73.

[2] J. Bengtsson, W. O. D. Griffioen, K. J. Kristoffersen, K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi, "Verification of an audio protocol with bus collision using UPPAAL," in *Computer Aided Verification, 8th International Conference, CAV '96, New Brunswick, NJ, USA, July 31 - August 3, 1996, Proceedings*, 1996, pp. 244–256.

[3] O. Maler and A. Pnueli, "Timing analysis of asynchronous circuits using timed automata," in *Correct Hardware Design and Verification Methods, IFIP WG 10.5 Advanced Research Working Conference, CHARME '95, Frankfurt/Main, Germany, October 2-4, 1995, Proceedings*, 1995, pp. 189–205.

[4] K. Huppler, "The art of building a good benchmark," in *Performance Evaluation and Benchmarking, First TPC Technology Conference, TPCTC 2009, Lyon, France, August 24-28, 2009, Revised Selected Papers*, 2009, pp. 18–30.

[5] D. Beyer, S. Löwe, and P. Wendler, "Reliable benchmarking: requirements and solutions," *International Journal on Software Tools for Technology Transfer*, Nov 2017.

[6] T. Tóth, A. Hajdu, A. Vörös, Z. Micskei, and I. Majzik, "Theta: a framework for abstraction refinement-based model checking," in *Proceedings of the 17th Conference on Formal Methods in Computer-Aided Design*, D. Stewart and G. Weissenbacher, Eds., 2017, pp. 176–179.

[7] J. Bengtsson and W. Yi, "Timed automata: Semantics, algorithms and tools," in *Lectures on Concurrency and Petri Nets*, ser. LNCS. Springer Berlin Heidelberg, 2004, vol. 3098, pp. 87–124.

[8] J. Bengtsson, K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi, "UPPAAL - a tool suite for automatic verification of real-time systems," in *Hybrid Systems III: Verification and Control, Proceedings of the DIMACS/SYCON Workshop on Verification and Control of Hybrid Systems, October 22-25, 1995, Ruttgers University, New Brunswick, NJ, USA*, 1995, pp. 232–243.

[9] R. Alur, T. A. Henzinger, and M. Y. Vardi, "Parametric real-time reasoning," in *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*, 1993, pp. 592–601.

[10] C. Baier and J. Katoen, *Principles of model checking*. MIT Press, 2008.

[11] F. Kordon and F. Hulin-Hubard, "Benchkit, a tool for massive concurrent benchmarking," in *14th International Conference on Application of Concurrency to System Design, ACSD 2014, Tunis La Marsa, Tunisia, June 23-27, 2014*, 2014, pp. 159–165.

[12] S. F. Arenis, B. Westphal, D. Dietsch, M. Muñiz, and A. S. Andisha, "The wireless fire alarm system: Ensuring conformance to industrial standards through formal verification," in *FM 2014: Formal Methods - 19th International Symposium, Singapore, May 12-16, 2014. Proceedings*, 2014, pp. 658–672.

[13] F. Herbreteau, B. Srivathsan, and I. Walukiewicz, "Lazy abstractions for timed automata," in *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, 2013, pp. 990–1005.

[14] C. Daws and S. Yovine, "Reducing the number of clock variables of timed automata," in *Proceedings of the 17th IEEE Real-Time Systems Symposium (RSS '96*. Washington - Brussels - Tokyo: IEEE, Dec. 1996, pp. 73–81.

[15] T. Tóth and I. Majzik, "Lazy reachability checking for timed automata using interpolants," in *Formal Modelling and Analysis of Timed Systems*, ser. Lecture Notes in Computer Science, A. Abate and G. Geeraerts, Eds. Springer, 2017, vol. 10419, pp. 264–280.