Controlling Timed Automata against MTL Specifications with TACoS

Till Hofmann[a], Stefan Schupp[b]

[a]*Knowledge-Based Systems Group, RWTH Aachen University, Aachen, Germany*
[b]*Cyber-Physical Systems Group, TU Wien, Vienna, Austria*

Abstract

TACoS is a tool for synthesizing controllers against specifications of undesired behavior with timing constraints. Given a timed automaton and an MTL specification, the tool synthesizes a controller that guarantees that every possible execution of the system satisfies the given specification. TACoS comes with a C++ library with a simple-to-use API and can read from and write to human-readable text input and output. In this paper, we outline the approach of the tool and present two examples in further detail.

Keywords: controller synthesis, timed automata, metric temporal logic

Nr.	**Code metadata description**	
C1	Current code version	1.2.0
C2	Permanent link to code/repository used for this code version	`https://doi.org/10.5281/zenodo.7194475`
C3	Permanent link to Reproducible Capsule	`https://doi.org/10.24433/CO.8796292.v1`
C4	Legal Code License	LGPL-3.0
C5	Code versioning system used	git
C6	Software code languages, tools, and services used	C++, cmake
C7	Compilation requirements, operating environments and dependencies	Linux OS, compiler which supports C++17 (gcc, clang), cmake ($\geq$ 3.14), boost, protobuf, graphviz
C8	Link to developer documentation/manual	`https://github.com/morxa/tacos/wiki` `https://morxa.github.io/tacos/`
C9	Support email for questions	`hofmann@kbsg.rwth-aachen.de` `stefan.schupp@tuwien.ac.at`

*© 2022. This manuscript version is made available under the CC-BY-NC-ND 4.0 license `https://creativecommons.org/licenses/by-nc-nd/4.0/`. The final authenticated version is available at `https://doi.org/10.1016/j.scico.2022.102898`.

1. Motivation

Digital systems interacting with the continuous real world are often safety critical in the sense that certain unwanted behavior should be avoided. Determining whether a given system is safe has been under extensive research for various system types. Related to the question of safety is the *synthesis of a controller*, which tries to find a component which can ensure safe execution of a system. Based on observations of the environment, the controller must select an action from a set of controllable actions such that the resulting execution trace is guaranteed to satisfy the formal specification.

For formal approaches towards the synthesis problem, an appropriate abstraction is required, which is able to reflect the relevant characteristics of a system without adding unnecessary information. Especially for systems with mixed discrete-continuous behavior, certain modelling concepts have been introduced. One of the most popular models are timed automata (TAs), which are widely used in the formal methods community to analyze formal safety properties.

While safety, i.e., the avoiding of certain system states, is one requirement that can be checked, other specifications such as reaching certain system states within a certain time interval (liveness) can be of interest. In general, specifications are usually formalized via logics. Here, we consider metric temporal logic (MTL), a timed extension of linear temporal logic (LTL), as formal specification language. Like LTL, MTL allows to describe temporal properties, but additionally allows timing constraints, e.g., $\mathbf{F}_{=1}a$, which requires that an a event occurs exactly one time unit in the future.

The tool TACoS implements an automated approach for the synthesis of TA controllers for a given MTL specification.

Related Work. Controller synthesis for timed systems has been researched extensively, in different settings. Tools such as Acacia+ [1], Unbeast [2], Strix [3], and Spot [4] synthesize controllers for LTL specifications. LTL synthesis has also been a focus of the Reactive Synthesis Competition (SYNTCOMP) [5]. In contrast to MTL, LTL does not allow timing constraints. SynthKro and FlySynth [6] synthesize controllers that remain in or reach a given set of states of a timed automaton rather than controlling against a logical specification. Uppaal-Tiga [7] and Synthia [8] control timed automata against a TCTL specification to accomplish reachability or safety. Uppaal-Tiga has also been extended to models with partial observability [9], using pre-defined controller templates. Casaal [10] synthesizes a controller for $\text{MTL}_{0,\infty}$ specifications. $\text{MTL}_{0,\infty}$ is a subset of MTL, where every bounded until operator may only use an upper or a lower time-bound

(not both). While many synthesis tools for different kinds of specifications exist, research on MTL synthesis so far has focussed on theoretical aspects. We intend to close this gap with TACoS, a tool for full MTL controller synthesis.

2. Software Description

The tool TACoS [11] is a tool for the automated synthesis of controllers for systems described by a timed automaton with respect to a specification formalized in MTL. As such, it augments well-known theoretical results [12] by a functional, easy-to-use implementation. Based on a TA representation of a system (the *plant*) and an MTL formula (the *specification*), TACoS synthesizes a TA which controls the plant against the specification such that the composition (i.e., the concurrent execution of the controller and the plant) does not violate the specification.

The implemented approach in TACoS is based on the theoretical contribution [12] but additionally focusses on practical applications. To realize this, the tool TACoS features multithreaded synthesis, various search heuristics, early termination, as well as smart pruning and re-using of the nodes in the search space. All methods required in the tool are shipped as C++ libraries to allow integration into other frameworks. Furthermore, we provide a parser for plant specifications based on Google's `protobuf`[1], as well as multiple visual output methods (interactive and non-interactive) to aid debugging.

3. Approach

We sketch the approach implemented in TACoS with a small example, where we have a timed automaton (see Figure 1) with one clock c. Initially, the automaton is in location l0, from which the clock can be reset at any non-zero (positive) time via the a-transition. Additionally, a second transition can be taken to reach the location l1, but only if one time unit has passed. The bad behavior is given by the MTL formula[2] $e \vee \mathbf{F}e$, which states that the system violates the specification if the e-transition to location l1 is taken.

The controller can only control the former transition (labelled with a), while the latter (e) is executed by the environment and thus is not controllable.

[1] `https://developers.google.com/protocol-buffers`

[2] We use MTL with strict semantics, where $\mathbf{F}\alpha$ is satisfied if α is satisfied in at any point strictly in the future. Therefore, $e \vee \mathbf{F}e$ is not equivalent to $\mathbf{F}e$.

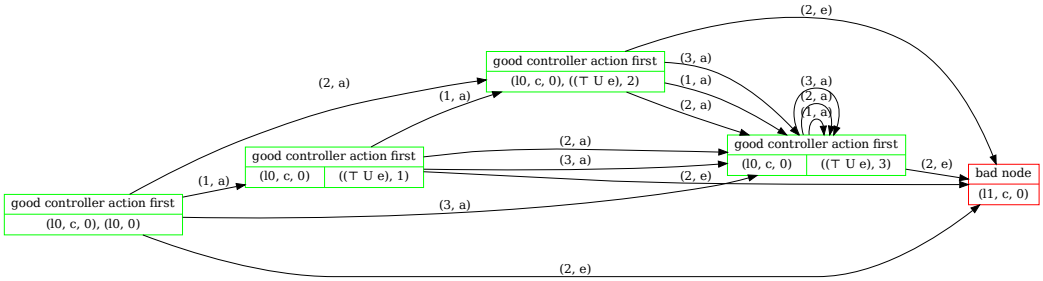


Figure 2: The search graph generated on the input TA from Figure 1 and the MTL specification $e \vee \mathbf{F}(e)$.

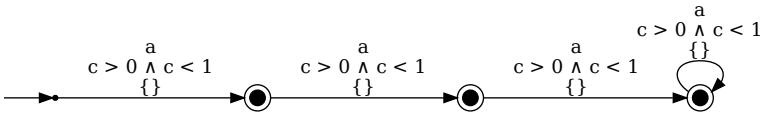


Figure 3: The synthesized controller.

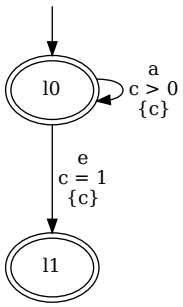


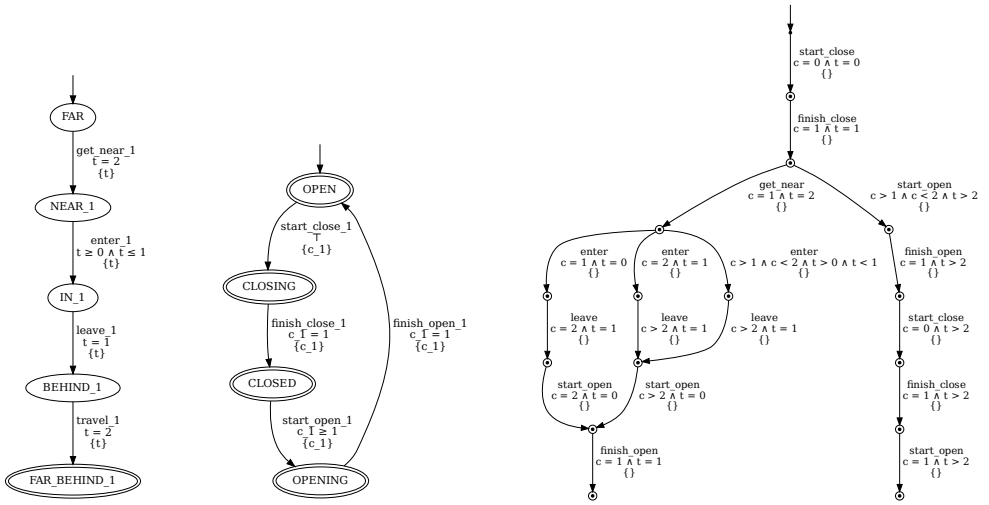
Figure 1: The plant.

To track the specification, the MTL formula is first translated to an alternating timed automaton (ATA) [13]. Intuitively, an ATA is an automaton that tracks the unsatisfied parts of the specification. During the search, a search graph (Figure 2) is explored, which tracks all possible evolutions of the plant along with the satisfaction status of the specification. A node in the search graph consists of time-abstracted TA and ATA configurations and therefore represents a set of of plant states which have similar discrete and temporal behavior. During search, the parts of the unwanted behavior (specification) that are not yet satisfied are updated based on the actions. Nodes that allow bad behavior, i.e., nodes where the specification of unwanted behavior is satisfied, are marked as *bad*, nodes which allow reaching such a node are subsequently labelled in case no controllable action allows to avoid reaching a bad node. A controller can be extracted from the search graph by avoiding nodes leading to bad behavior (see Figure 3). The resulting controller is again a TA that is executed in parallel to the plant. As the controller must not limit the acceptance behavior of the plant, all of its locations are accepting (double circle in Figure 3), such that plant states that are accepting are also accepting in the composition of plant and controller.

As shown in [12], the search always terminates and the resulting controller is correct. The controller is in general not unique, but in this case implements a very simple strategy of taking the a-transition repeatedly, thus prohibiting the environment from executing its transition (e) which potentially leads to

bad behavior.

4. Example



(a) The train. (b) The gate. (c) The resulting controller that controls the gate against the specification.

Figure 4: A railroad crossing modelled as a synthesis problem: The plant consists of two TAs: the *train* and the *gate*. While the train's actions are controlled by the environment, the gate's actions are controllable. The undesired behavior is specified as $((\mathit{enter} \vee \mathbf{F}_{[0,1]}\,\mathit{enter})\,\widetilde{\mathbf{U}}\,\neg \mathit{finish\_close}) \vee (\mathit{start\_open}\,\widetilde{\mathbf{U}}\,\neg \mathit{leave}) \vee (\mathit{travel}\,\widetilde{\mathbf{U}}\,\neg \mathit{finish\_open})$, which intuitively expresses that (1) the gate must be closed for at least 1 sec before the train may enter, (2) the gate must not open before the train has left the crossing, (3) the gate must open before the train is far behind the gate.

We consider a classical example of a train-gate system [14]. As shown in Figure 4, the system consists of two parts: an uncontrollable train and a controllable gate. The train in Figure 4a approaches the gate and is modelled as TA with some timing constraints on its actions, which describe the travelling time of the train. The gate is modelled as a second TA that may open and close, as shown in Figure 4b. Opening and closing both takes some time, which is modeled by having the intermediate states *CLOSING* and *OPENING* and timing constraints on the finish actions. The goal is to synthesize a controller that controls the gate such that the gate is always closed when the train passes the crossing. Additionally, we require the gate to be closed for at least 1 sec before the train may enter the crossing. Formally,

the undesired behavior can be expressed with the following MTL formula:

$$((\mathit{enter} \vee \mathbf{F}_{[0,1]}\,\mathit{enter})\,\widetilde{\mathbf{U}}\,\neg \mathit{finish\_close}) \\ \vee (\mathit{start\_open}\,\widetilde{\mathbf{U}}\,\neg \mathit{leave}) \vee (\mathit{travel}\,\widetilde{\mathbf{U}}\,\neg \mathit{finish\_open})$$

Figure 4c shows the synthesized controller. Note that the controller is able to react to every possible (modelled) behavior of the environment, e.g., *enter* actions at different timepoints. Additionally, if the train never enters the gate, any sequence of gate operations is admissible because the system will never reach a final state (which requires that the train is in the location *FAR_BEHIND_1*). This can be seen in the right branch of the controller: If the *get_near* action does not occur, then the controller responds with a sequence of start/open actions. While these actions do not violate the specification, they are not necessary. As this example shows, the resulting controller is not necessarily minimal.

5. Impact

TACoS provides the first implementation of TA controller synthesis against full MTL specifications. It ships with a C++ API, which allows applications to integrate the synthesis library, as well as a protobuf API, which can be used for textual input in a standalone binary.

In the current implementation, a TA controller is synthesized, but we aim to generalize the approach. We especially expect TACoS to serve as a basis for further development in the area of program synthesis in the context of robotics, which is part of the current development.

Acknowledgements

This work was funded by the German Research Council (DFG) in the context of the research training group UnRAVeL (RTG 2236), DFG grant GL-747/23-1 ConTrAkt, and the EU ICT-48 2020 project TAILOR (No. 952215).

References

[1] A. Bohy, V. Bruyère, E. Filiot, N. Jin, J.-F. Raskin, Acacia+, a Tool for LTL Synthesis, in: Computer Aided Verification, Springer, 2012, pp. 652–657.

[2] R. Ehlers, Unbeast: Symbolic Bounded Synthesis, in: Tools and Algorithms for the Construction and Analysis of Systems, Springer, 2011, pp. 272–275.

[3] P. J. Meyer, S. Sickert, M. Luttenberger, Strix: Explicit Reactive Synthesis Strikes Back!, in: Computer Aided Verification, Springer International Publishing, 2018, pp. 578–586.

[4] A. Duret-Lutz, A. Lewkowicz, A. Fauchille, T. Michaud, É. Renault, L. Xu, Spot 2.0 — A Framework for LTL and $$\omega $$-Automata Manipulation, in: Automated Technology for Verification and Analysis, Springer International Publishing, 2016, pp. 122–129.

[5] S. Jacobs, G. A. Perez, R. Abraham, V. Bruyere, M. Cadilhac, M. Colange, C. Delfosse, T. van Dijk, A. Duret-Lutz, P. Faymonville, B. Finkbeiner, A. Khalimov, F. Klein, M. Luttenberger, K. Meyer, T. Michaud, A. Pommellet, F. Renkin, P. Schlehuber-Caissier, M. Sakr, S. Sickert, G. Staquet, C. Tamines, L. Tentrup, A. Walker, The Reactive Synthesis Competition (SYNTCOMP): 2018-2021 (Jun. 2022). arXiv:2206.00251.

[6] K. Altisen, S. Tripakis, Tools for Controller Synthesis of Timed Systems, in: Proceedings of the 2nd Workshop on Real-Time Tools, 2002, pp. 1–12.

[7] G. Behrmann, A. Cougnard, A. David, E. Fleury, K. G. Larsen, D. Lime, UPPAAL-Tiga: Time for Playing Games!, in: Computer Aided Verification, Springer, 2007, pp. 121–125.

[8] H.-J. Peter, R. Ehlers, R. Mattmüller, Synthia: Verification and Synthesis for Timed Automata, in: Computer Aided Verification, Springer, 2011, pp. 649–655.

[9] B. Finkbeiner, H.-J. Peter, Template-Based Controller Synthesis for Timed Systems, in: Tools and Algorithms for the Construction and Analysis of Systems, Springer, 2012, pp. 392–406.

[10] G. Li, P. G. Jensen, K. G. Larsen, A. Legay, D. B. Poulsen, Practical controller synthesis for MTL0,∞, in: Proceedings of the 24th ACM SIGSOFT International SPIN Symposium on Model Checking of Software, Association for Computing Machinery, 2017, pp. 102–111.

[11] T. Hofmann, S. Schupp, TACoS: A tool for MTL controller synthesis, in: R. Calinescu, C. S. Păsăreanu (Eds.), Software Engineering and Formal Methods, Springer International Publishing, Cham, 2021, pp. 372–379. doi:10.1007/978-3-030-92124-8_21.

[12] P. Bouyer, L. Bozzelli, F. Chevalier, Controller Synthesis for MTL Specifications, in: Proceedings of the 17th International Conference on Concurrency Theory (CONCUR), Springer Berlin Heidelberg, 2006, pp. 450–464.

[13] J. Ouaknine, J. Worrell, On the decidability of metric temporal logic, in: 20th Annual IEEE Symposium on Logic in Computer Science (LICS' 05), 2005, pp. 188–197.

[14] R. Alur, T. A. Henzinger, M. Y. Vardi, Parametric real-time reasoning, in: Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, Association for Computing Machinery, 1993, pp. 592–601.