TACoS: A Tool for MTL Controller Synthesis

Till Hofmann[1] and Stefan Schupp[2]

[1] Knowledge-Based Systems Group, RWTH Aachen University, Aachen, Germany
hofmann@kbsg.rwth-aachen.de
[2] Cyber-Physical Systems Group, TU Wien, Vienna, Austria
stefan.schupp@tuwien.ac.at

Abstract. We introduce TACoS, a tool for synthesizing controllers satisfying MTL specifications of undesired behavior with timing constraints. Our contribution extends an existing theoretical approach towards practical applications. The most notable features include: Online labeling to terminate early if a solution has been found, heuristic search to expand the most promising nodes first, search graph pruning to reduce the problem size by pruning irrelevant parts of the search graph, and re-using previously explored search nodes to further reduce the search graph. Finally, multi-threading support allows to make use of modern CPUs with many parallel threads. TACoS comes with a C++ library with minimal external dependencies and simple-to-use API. We evaluate our approach on a number of scenarios and investigate how each of the enhancements improves the performance.
The tool is publicly available at https://github.com/morxa/tacos.

Keywords: controller synthesis · metric temporal logic.

1 Introduction

Controller synthesis is the problem of determining a controller for a given system to ensure the behavior of the composed system follows a certain specification. The problem has been researched extensively for different kinds of systems and different kinds of specifications (e.g., [4,9,7]). It has also seen interest in the AI community (e.g., [8]) and in robotics (e.g., [15,12,13]). One particular synthesis problem is *controller synthesis for MTL specifications* [7], where the system is modeled as timed automaton (TA) and the specification of undesired behavior is given as a metric temporal logic (MTL) formula. The problem has shown to be decidable for finite words and fixed resources [7]. While several applications are based on metric temporal constraints (e.g., [17,20,22]), to the best of our knowledge, no general implementation of such a synthesis approach exists.

Supported by DFG RTG 2236 *UnRAVeL* and DFG grant GL-747/23-1 *ConTrAkt*.
This is a preprint version. The final authenticated version is available online at https://doi.org/10.1007/978-3-030-92124-8_21.

Related Work Controller synthesis for timed systems has been researched extensively, in different settings. Tools such as Acacia+ [6] and Unbeast [10] synthesize controllers for LTL specifications, which does not allow time constraints. SynthKro and FlySynth [1] synthesize controllers that remain in or reach a given set of states of a timed automaton. Uppaal-Tiga [5] and Synthia [19] control timed automata against a TCTL specification to accomplish reachability or safety. Uppaal-Tiga has also been extended to models with partial observability [11], using pre-defined controller templates. Casaal [16] synthesizes a controller for $\mathrm{MTL}_{0,\infty}$ specifications. $\mathrm{MTL}_{0,\infty}$ is a subset of MTL, where every bounded until operator may only use an upper or a lower time-bound (not both).

In this work, we present TACoS, a *TA Controller Synthesis* tool for MTL specifications, based on theoretical decidability results from [7]. In Section 2, we summarize the MTL synthesis problem, before we describe our tool in more detail in Section 3. In Section 4, we evaluate TACoS on benchmarks from several scenarios, before we conclude in Section 5.

2 The MTL Synthesis Problem

Timed automata (TA) [2] are a widely used model for representing real-timed and hybrid systems. Their properties are often described with MTL [14], a temporal logic that extends linear temporal logic (LTL) with metric time on the *Until* modality. One commonly used semantics for MTL is a *pointwise semantics*, in which formulas are interpreted over timed words. A timed word ρ over a finite set of atomic propositions AP is a finite or infinite sequence $\rho = (\sigma_0, \tau_0)\,(\sigma_1, \tau_1)\ldots$ where $\sigma_i \in AP$ and $\tau_i \in \mathbb{R}_+$ such that the sequence (τ_i) is monotonically non-decreasing and non-Zeno. We use $|\rho|$ to denote the number of elements in ρ. For a set AP of atomic propositions, the formulas of MTL are built from $\phi ::= a \mid \neg\phi \mid \phi \wedge \phi \mid \phi\,\mathbf{U}_I\,\phi$ (where $a \in AP$). We use the short-hand notations $\phi\,\widetilde{\mathbf{U}}_I\,\psi := \neg\,(\neg\phi\,\mathbf{U}_I\,\neg\psi)$ (*dual until*), $\mathbf{F}_I\phi := (\top\,\mathbf{U}_I\,\phi)$ (*finally*) and $\mathbf{G}_I\phi := \neg\mathbf{F}_I\neg\phi$ (*globally*). Given a timed word $\rho = (\sigma_0, \tau_0)\,(\sigma_1, \tau_1)\ldots$ over alphabet AP and an MTL formula ϕ, $\rho, i \models \phi$ is defined as usual for the boolean operators, and with the following rule for $\mathbf{U}_I$: $\rho, i \models \phi_1\,\mathbf{U}_I\,\phi_2$ iff there exists j such that (1) $i < j < |\rho|$, (2) $\rho, j \models \phi_2$, (3) $\tau_j - \tau_i \in I$, (4) and $\rho, k \models \phi_1$ for all k with $i < k < j$. We also write $\rho \models \phi$ for $\rho, 0 \models \phi$ and we define the language of ϕ as $L(\phi) = \{\rho \mid \rho \models \phi\}$.

MTL Control Problem. The goal is to synthesize a controller $\mathcal{C}$ that *controls* a *plant* $\mathcal{P}$ against a specification of undesired behaviors Φ such that all resulting traces in the composition of $\mathcal{P}$ and $\mathcal{C}$ satisfy the specification Φ without blocking the plant $\mathcal{P}$. In this context, *control* means that $\mathcal{C}$ has control over some actions, while the environment controls the remaining actions. The synthesis problem on finite words and finite resources (i.e., fixed number of clocks and fixed constants) is decidable [7]. We refer to [7] for the formal definition.

3 Approach

Based on [7], our tool works as follows: First, it translates the specification into an alternating timed automaton (ATA) [18]. Next, it recursively constructs a tree over regionalized configurations of the synchronous product of the plant TA $\mathcal{A}$ and the specification ATA $\mathcal{B}$. Intuitively, each node n in the search tree contains a single regionalized configuration $n_{\mathcal{A}}$ of $\mathcal{A}$ and a set $n_{\mathcal{B}}$ of possible configurations of $\mathcal{B}$, which represents parts of the specification that have not been satisfied yet. Each newly discovered node in the search tree is *expanded* by computing all (regionalized) time and jump successors $n'_{\mathcal{A}}$ of $n_{\mathcal{A}}$ and the respective $N'_{\mathcal{B}}$ for all symbols. Nodes in which the $\mathcal{A}$ configuration is in a final location and Φ has fully been satisfied ($\mathcal{B}$ is accepting) are labeled as *bad*, as they represent cases in which the plant is in a final state and the specification has been violated. After building the search tree, the tree is traversed and labeled bottom-up (*good*, *bad*) based on the labels of the leaf nodes. A controller exists if the root node is labeled *good*.

TACoS aims to provide a practicable tool to synthesize TA controllers against an MTL specification, with a focus on performance and usability. We summarize the most notable features in the following.

Parallelization. To make use of multi-threading, the node expansion is parallelized. Pending nodes are stored in a globally accessible queue and worker threads take nodes from the queue, expand those and push resulting successors into the queue for further processing.

Incremental labeling. Instead of first constructing a complete search tree and then labeling the tree bottom-up, it is also possible to partially label the tree during expansion. Nodes are labelled recursively until either the root node has been labeled or not enough information is available to label a node. This approach allows to label the root node without constructing the complete search tree.

Pruning. With incremental labeling, a node's label may be determined during search. With pruning, whenever a node's label is determined, all of its unlabeled successors are marked as *canceled*, which prevents them from being expanded later on. The combination of incremental labeling and pruning allows to effectively skip large parts of the search graph during construction.

Node re-using. When constructing the search tree as described, many nodes are created multiple times. This may occur whenever certain states of the system are reachable via different execution paths of the plant. Duplicate nodes consequently agree on their subtrees, i.e., the work of exploring these subtrees will be done several times. To overcome this, we identify duplicate nodes during the search. Instead of re-creating the sub-tree, we re-use the existing node instead and add the corresponding edges. This changes the underlying data structure from a search tree to a *search graph*, affecting all other improvements as well.

Search heuristics. Incremental labeling and search-graph pruning heavily depend on the order in which nodes are expanded. We provide several heuristics which determine the order of nodes in the queue: breadth-first-search (`bfs`) and depth-first-search (`dfs`) work as expected. A heuristics based on timing (`time`) prioritizes the node with the shortest accumulated time (global time). The heuristic `cw` prefers nodes with configurations where more parts of the specification (of undesired behavior) are not yet satisfied. The heuristic `env` prefers environment actions over controller actions, based on the intuition that the controller should only act if necessary (and let the plant run otherwise). The composite heuristic `comp` is a weighted sum of other heuristics. In the following, we have used $\texttt{comp} = 16 \cdot \texttt{cw} + 4 \cdot \texttt{env} + 1 \cdot \texttt{time}$. Finally, the tool also provides a `random` heuristic, which is mainly helpful for comparison and testing.

Action- and location-based specification. The approach in [7] suggests a method designed for *action-based* specifications in which labels on transitions (the *actions*) are used. However, *location-based* specifications, which specify the desired or undesired behavior in terms of properties on locations, sometimes allow a more intuitive specification. Our tool supports both types of specifications.

Utility. To ease debugging, we provide several utility functions such as plotting of the input automata, the resulting controller, or the search graph. TACoS reads text input and is shipped with a C++ library with simple API to create input programmatically. The synthesis result can be stored in a human-readable or binary format. TACoS can also be run in an interactive mode after search, which allows to debug the controller synthesis step-by-step with visual support.

4 Evaluation

We evaluated our system on several scenarios and ran each scenario in each configuration five times. All experiments were conducted on an AMD Ryzen 7 3700X with 16 parallel threads and 32 GB memory. We measured the number of locations in the input problem, the number of nodes in the search graph, the number of explored nodes in the search graph, and the number of locations in the resulting controller. We used three scenarios:

Example 1 (Railroad). This is a variant of the train-gate controller [3]. A train approaches a crossing, the controller needs to open and close the gate such that the train can pass. The problem is modeled as product of two TAs. The train performs the uncontrollable actions *get_near, enter, leave, travel* in sequence, i.e., approaches and passes through the gate section. The gate may perform the controllable actions *start_close,start_open, finish_close, finish_open* to change its state. The composed system is safe if the gate is closed when the train enters and opens after the train leaves the crossing. Thus, the bad behavior is defined by

$$\mathit{enter}\,\widetilde{\mathbf{U}}\,\neg\mathit{finish\_close} \lor \mathit{start\_open}\,\widetilde{\mathbf{U}}\,\neg\mathit{leave} \lor \mathit{travel}\,\widetilde{\mathbf{U}}\,\neg\mathit{finish\_open}$$

Table 1. A comparison of the heuristics implemented in TACoS for an instance of the railroad example. We compare the used heuristics (heu), the resulting running time (wall) and CPU time (CPU) in seconds, the size of the search tree (nodes) and the number of explored nodes (expl) in thousands of nodes as well as the number of locations in the resulting controller (ctrl). Standard deviations are given in brackets, e.g., 1.1(2) means 1.1 ± 0.2.

Scenario	size	heu	wall (s)	CPU (s)	nodes (k)	expl (k)	ctrl
Railroad(2,2)	144	bfs	53.9(9)	53.8(9)	18.32(2)	7.8(2)	53(7)
		dfs	11(4)	11(4)	12(2)	8(1)	79(29)
		cw	8(3)	8(3)	10(2)	6.0(9)	71(8)
		env	11(3)	11(3)	11(2)	3.3(8)	46(3)
		time	65.6(9)	65.5(9)	17.99(8)	3.09(7)	52(10)
		rand	13(5)	13(5)	13(2)	2.7(6)	71(20)
		comp	4(3)	4(3)	6(3)	4(2)	32(10)

We have parameterized the problem by the number of crossings and the distances before each crossing, where `Railroad(4,8)` is the problem with two crossings and a distance of 4 and 8 time units before the first and second crossing.

Example 2 (Robot). A robot transports goods between stations (based on [22]). It has a camera that needs to be enabled 1 sec before the robot performs a *pick* or a *put* action. As the camera may overheat, it must not run continuously for longer than 4 sec. The camera is controllable with the actions *on* and *off*, the robot's actions *pick*, *put*, and *move* are not controllable. The robot takes exactly 3 sec to move between the stations. The specification of undesired behavior is given as:

$$\begin{aligned}&\neg \mathit{on}\,\mathbf{U}\,\mathit{pick} \lor \mathbf{F}(\mathit{off} \land (\neg \mathit{on}\,\mathbf{U}\,\mathit{pick})) \lor \mathbf{F}\big(\mathit{on}\,\mathbf{U}_{[0,1]}\,\mathit{pick}\big)\\ &\lor \neg \mathit{on}\,\mathbf{U}\,\mathit{put} \lor \mathbf{F}(\mathit{off} \land (\neg \mathit{on}\,\mathbf{U}\,\mathit{put})) \lor \mathbf{F}\big(\mathit{on}\,\mathbf{U}_{[0,1]}\,\mathit{put}\big)\end{aligned}$$

Example 3 (Conveyor Belt). A conveyor belt moves luggage in an airport (based on [21]). If a piece of luggage gets stuck, the belt must stop, which allows the luggage to be removed. The conveyor must not immediately continue but instead wait for at least 2 sec. Also, the conveyor should not stop without reason. The controllable actions are *move* and *stop*, while the uncontrollable actions are *release*, *resume*, and *stuck*. The undesired behavior is specified as follows:

$$\mathbf{F}\big(\mathit{release} \land \mathbf{F}_{[0,2]}\,\mathit{move}\big) \lor (\neg \mathit{stuck})\ \mathbf{U}\,\mathit{stop} \lor \mathbf{F}(\mathit{stop} \land (\neg \mathit{stuck})\ \mathbf{U}\,\mathit{stop})$$

Results We first compare the different heuristics in Table 1. We can see that using heuristics is generally helpful and improves both the running time and the resulting search size and controller size when compared to `bfs`. Interestingly, the heuristic `time` does not perform well and is actually worse than `bfs`, `dfs`, and even `random`. Also, `dfs` performs surprisingly well compared to the other heuristics, at least in this scenario. With some margin, the composite heuristic

Table 2. A comparison of single- and multi-threading (with 16 threads). We compare the used heuristics (heu), whether multi-threading is used (multi), the resulting running time (wall) and CPU time (CPU) in seconds, the size of the search tree (nodes) and the number of explored nodes (expl) in thousands of nodes as well as the number of locations in the resulting controller (ctrl). Standard deviations are given in brackets, e.g., 24(3) means 24 ± 3.

Scenario	heu	multi	wall (s)	CPU (s)	nodes (k)	expl (k)	ctrl
Railroad(2,2)	comp	n	4(3)	4(3)	6(3)	4(2)	32(10)
	comp	y	1.2(4)	10(4)	9(2)	6(2)	60(27)
Robot	comp	n	0.289(2)	0.289(2)	0.182(1)	0.067(9)	30(7)
	comp	y	0.134(3)	0.66(1)	0.40(4)	0.065(8)	34.4(5)
Conveyor	comp	n	0.44(7)	0.44(7)	0.45(2)	0.35(5)	150(33)
	comp	y	0.37(2)	0.57(2)	0.46(1)	0.37(2)	166(5)

Table 3. The *railroad* problem scaled to different travel times and number of crossings, using the `comp` heuristic and multi-threading. We provide the size of the timed automaton (size), the resulting running time (wall) and CPU time (CPU) in seconds, the size of the search tree (nodes) and the number of explored nodes (expl) in thousands of nodes as well as the number of locations in the resulting controller (ctrl). Standard deviations are given in brackets, e.g., 0.14(5) means 0.14 ± 0.05.

Scenario	size	wall (s)	CPU (s)	nodes (k)	expl (k)	ctrl
Railroad(2,2)	144	1.2(1)	10(1)	9.3(7)	5.8(5)	43(5)
Railroad(2,4)	144	4.2(4)	41(8)	22.6(7)	14(2)	49(9)
Railroad(2,8)	144	20(7)	211(103)	58(7)	31(8)	47(10)
Railroad(4,4)	144	11.4(7)	151(13)	32.4(1)	22.38(8)	48(2)
Railroad(4,8)	144	63(12)	909(192)	83(6)	51(5)	64(19)
Railroad(8,8)	144	280(93)	4313(1506)	111(3)	75(1)	45(10)
Railroad(1,1,1)	832	35(11)	380(138)	134(39)	62(12)	74(82)
Railroad(2,1,1)	832	18 768(2868)	298 585(45821)	448(34)	333(21)	101(31)
Railroad(2,2,2)	832	36 537(12434)	582 279(198204)	493(129)	368(72)	103(45)

`comp` performs best. Second, we evaluate multi-threaded search, running times are shown in Table 2. We can see that multi-threading reduces the running time, but increases CPU time and often has a negative impact on search size and controller size, most likely as additional nodes with a worse heuristic value are expanded as well when computing with multiple threads. Finally, Table 3 shows the performance on the scaled railroad problem. We can see that TACoS is able to find a controller even for large input problems, although the running time increases significantly. Further results are available on the tool webpage[3].

[3] https://github.com/morxa/tacos

5 Conclusion

We have presented TACoS, to our knowledge the first tool for TA controller synthesis against MTL specifications. TACoS comes with a number of features aiming to provide both good performance and usability. We have evaluated the tool in three settings, which showed that it is capable of synthesizing controllers with reasonable performance. To further improve its performance, investigating more sophisticated heuristics would be a promising next step. Also, in future work, we want to investigate the applicability of the presented approach for control program synthesis and its performance on real robotic systems.

References

1. Altisen, K., Tripakis, S.: Tools for Controller Synthesis of Timed Systems. In: RT-TOOLS (2002)
2. Alur, R., Dill, D.: A theory of timed automata. TCS **126**(2) (1994)
3. Alur, R., Henzinger, T., Vardi, M.: Parametric real-time reasoning. In: STOC (1993)
4. Asarin, E., Maler, O., Pnueli, A., Sifakis, J.: Controller synthesis for timed automata. IFAC **31**(18) (1998)
5. Behrmann, G., Cougnard, A., David, A., Fleury, E., Larsen, K.G., Lime, D.: UPPAAL-Tiga: Time for Playing Games! In: CAV (2007)
6. Bohy, A., Bruyère, V., Filiot, E., Jin, N., Raskin, J.F.: Acacia+, a Tool for LTL Synthesis. In: CAV (2012)
7. Bouyer, P., Bozzelli, L., Chevalier, F.: Controller synthesis for MTL specifications. In: CONCUR (2006)
8. De Giacomo, G., Vardi, M.: Synthesis for LTL and LDL on finite traces. In: IJCAI (2015)
9. D'souza, D., Madhusudan, P.: Timed control synthesis for external specifications. In: STACS (2002)
10. Ehlers, R.: Unbeast: Symbolic Bounded Synthesis. In: TACAS (2011)
11. Finkbeiner, B., Peter, H.J.: Template-Based Controller Synthesis for Timed Systems. In: TACAS (2012)
12. He, K., Lahijanian, M., Kavraki, L., Vardi, M.: Reactive synthesis for finite tasks under resource constraints. In: IROS (2017)
13. Hofmann, T., Lakemeyer, G.: Controller synthesis for Golog programs over finite domains with metric temporal constraints. arXiv:2102.09837 (2021)
14. Koymans, R.: Specifying real-time properties with metric temporal logic. Real-Time Systems **2**(4) (1990)
15. Kress-Gazit, H., Fainekos, G., Pappas, G.: Temporal-logic-based reactive mission and motion planning. IEEE Transactions on Robotics **25**(6) (2009)
16. Li, G., Jensen, P.G., Larsen, K.G., Legay, A., Poulsen, D.B.: Practical controller synthesis for MTL0,∞. In: SPIN (2017)
17. Nikou, A., Tumova, J., Dimarogonas, D.: Cooperative task planning of multi-agent systems under timed temporal specifications. In: ACC (2016)
18. Ouaknine, J., Worrell, J.: On the decidability of metric temporal logic. In: LICS (2005)
19. Peter, H.J., Ehlers, R., Mattmüller, R.: Synthia: Verification and Synthesis for Timed Automata. In: CAV (2011)

20. Saha, S., Julius, A.: An MILP approach for real-time optimal controller synthesis with Metric Temporal Logic specifications. In: ACC (2016)
21. van Hulst, A., Reniers, M., Fokkink, W.: Maximally permissive controlled system synthesis for non-determinism and modal logic. DEDS **27**(1) (2017)
22. Viehmann, T., Hofmann, T., Lakemeyer, G.: Transforming robotic plans with timed automata to solve temporal platform constraints. In: IJCAI (2021)