Demo Abstract: Temporal Behavior Trees – Segmentation

Sebastian Schirmer
Johann C. Dauer
first.lastname@dlr.de
DLR German Aerospace Center
Braunschweig, Germany

Jasdeep Singh
Emily Jensen
Sriram Sankaranarayanan
first.lastname@colorado.edu
University of Colorado Boulder
Boulder, USA

Bernd Finkbeiner
finkbeiner@cispa.de
CISPA Helmholtz Center for Information Security
Saarbrücken, Germany

ABSTRACT

We present our tool for the segmentation of *temporal behavior trees* (TBT), a novel formalism for monitoring specifications. TBTs can be easily retrofitted to behavior trees, commonly used to program robotic applications. Our tool supports the robustness semantics of TBT and generates trace segmentations. In other words, given a TBT specification and a trace, it determines the optimal assignment of TBT nodes to sub-traces. To illustrate its application, we use the example of an autonomous ship deck landing. We showcase the user inputs required and demonstrate how the outputs can be interpreted to identify challenging task aspects, contributing to a comprehensive system analysis.

KEYWORDS

CPS, Offline segmentation, temporal behavior trees, temporal logic

ACM Reference Format:
Sebastian Schirmer, Johann C. Dauer, Jasdeep Singh, Emily Jensen, Sriram Sankaranarayanan, and Bernd Finkbeiner. 2024. Demo Abstract: Temporal Behavior Trees – Segmentation. In *27th ACM International Conference on Hybrid Systems: Computation and Control (HSCC '24), May 14–16, 2024, Hong Kong, Hong Kong*. ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3641513.3652534

1 MOTIVATION

Behavior trees (BT) [1] are a popular mathematical model to program robotic applications that represent a plan to execute behaviors. They consist of tree nodes that specify a sequence of sub-plans (Seq); falling back to a different sub-plan if the current sub-plan fails (Fback); and conducting many sub-plans in parallel until the number of sub-plans that have succeeded exceeds a specified lower limit [1]. In Figure 1, we present a BT that facilitates two different landing maneuvers for an unmanned aerial vehicle (UAV) on a ship deck. The BT starts with the *Straight-In* maneuver, involving the UAV moving behind the ship (1) and then maintaining this position for a designated time (3). If either (1) or (3) fails, the UAV switches to the *45-Degree* maneuver, encompassing diagonal movement behind the ship (2) and then maintaining this position for a designated time (4). Following the successful completion of either (3) or (4), the execution proceeds with moving above the touchdown point (5), before then descending (6). Despite the prevalence of BTs for *executing* behaviors, our novel formalism temporal behavior trees (TBT) fills the gap of providing a native and straightforward formalism to *monitor* behavior executions.

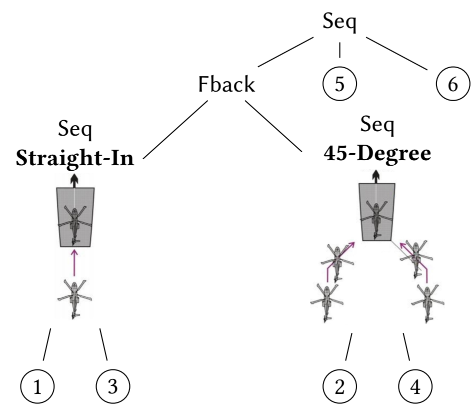

Figure 1: Behavior tree with two maneuvers to land on a ship.

2 TEMPORAL BEHAVIOR TREE

Temporal behavior trees (TBT) retrofit monitoring properties to BT by specifying temporal properties at their leaf nodes and providing a formal semantics for BT operators. The syntax, semantics, and the technical details, such as algorithms to monitor and segment TBT over traces, are provided in our companion paper that accompanies this tool description [4]. Our tool[2] supports the robust semantics of signal temporal logic (STL) to specify temporal properties on the leaf nodes [2, 3]. Thus, a formula yields a numerical value upon evaluation, with a positive value indicating satisfaction and a negative value indicating violation. Further, the numerical value is proportional to the degree of satisfaction. Figure 2 provides temporal formulas for each leaf node in Figure 1. The formulas (1) / (2) and (3) / (4) differ in their target position that should eventually be reached. The information received from the system are the position of the UAV (p_{uav}) and the ship (p_s), as well as their respective heading h_{uav} and h_s. The variable *aligned* is a constant and $p_{touchdown}$ is computed based on p_s. It's noteworthy that a fundamental characteristics of TBT is that the user does not directly dictate the transition from one leaf node to the next. Instead, this transition decision becomes the responsibility of an algorithm utilizing TBT as a specification language.

[1] www.behaviortree.dev

[2] https://github.com/DLR-FT/TBT-Segmentation

HSCC '24, May 14–16, 2024, Hong Kong, Hong Kong

ACM ISBN 979-8-4007-0522-9/24/05
https://doi.org/10.1145/3641513.3652534

Leaf	Temporal Formula
1	$\Diamond \ behind(p_s, p_{uav})$
2	$\Diamond \ diagonalBehind(p_s, p_{uav})$
3	$\Box_{[0,5]} \ (behind(p_s, p_{uav}) \ \wedge \ heading(aligned, h_s, h_{uav}))$
4	$\Box_{[0,5]} \ (diagBehind(p_s, p_{uav}) \ \wedge \ heading(aligned, h_s, h_{uav}))$
5	$\Diamond \ (move\_touchdown(p_s, p_{uav}) \ \wedge \ heading(aligned, h_s, h_{uav}))$
6	$\Diamond \ (descended(p_{touchdown}, p_{uav}))$

Figure 2: TBT that retrofits formulas to the BT in Figure 1.

3 SEGMENTATION

The first algorithm that uses TBT as specification language is an offline algorithm to segment traces [4]: Given a TBT and a trace of a system, the algorithm optimally assigns nodes of the TBT to subtraces. Figure 3 visualizes such a computed segmentation. Dotted lines represents the best positions relative to the ship. The numbers indicate the corresponding leaf nodes in Figure 2. Compared to other monitoring approaches [2, 3], which compute a single verdict for the whole trace, our tool decides optimal transitions between tree nodes and provides a satisfaction verdict for each segment.

4 TOOL DESCRIPTION FOR SEGMENTATION

The TBT monitoring, robustness, and segmentation toolbox is written in Rust. It requires the user to implement two functions:

(1) **fn** get_trace(...) -> Trace that produces a Trace, e.g., by reading and parsing a CSV-logfile, and
(2) **fn** get_tree(...) -> Tbt that constructs the TBT specification as a syntax tree.

To run the application, we build it after defining the two functions above and input a logfile representing the trace:

```
tbt-segmentation --logfile <your-logfile-location>
```

The optional flags --sampling and --lazy enable the use of the stuttering reduction and lazy computation as described in our paper [4]. By default, the tool provides information for all nodes in the TBT. For instance, there is one line in the output for each TBT node similar to: "lower: 0 upper: 506 value: 0.49829227 segment: Seq(22)" where lower represents the starting index of the TBT node in the logfile, upper the ending index, value represents its robustness value, segment represents the corresponding node. Thus, the line corresponds to the node with ID 22 – the root node in Figure 1. The root node was assigned to the segment that starts at Index 0 in the logfile and ends at Index 506. Since its value is positive, the segment satisfies the TBT. Other similar formatted output lines that correspond to leaf nodes further refine the analysis and help to identify the most challenging segments, i.e., which leaf node contributes the value. By providing --children as command-line-argument only leaf information are printed.

By default, not only the optimal segmentation but also three alternative segmentations are computed. This can be changed by --amount <number>. Further, the search for alternative segmentations can be regulated through the parameters --tau <number> and --rho <number>. These parameters specify that in an alternative segmentation, the segments must differ by at least tau in their starting and ending indices. Additionally, at least the robustness of one segment must differ by rho. These constraints are useful since otherwise already minor changes, e.g., changing the upper value of Seq(22) from 506 to 505, would result in a valid alternative segmentation. Note that if a different leaf node is utilized or the order of leaf nodes are altered in an alternative segmentation, then both conditions are satisfied. Therefore, by selecting a parameter that is "unachievable", such as setting tau to a value greater than the trace length, an alternative segmentation must opt for distinct leaf nodes. In our example, if the optimal segmentation corresponds to a *45-Degree* maneuver, the alternative must then be a segmentation using the *Straight-In* maneuver.

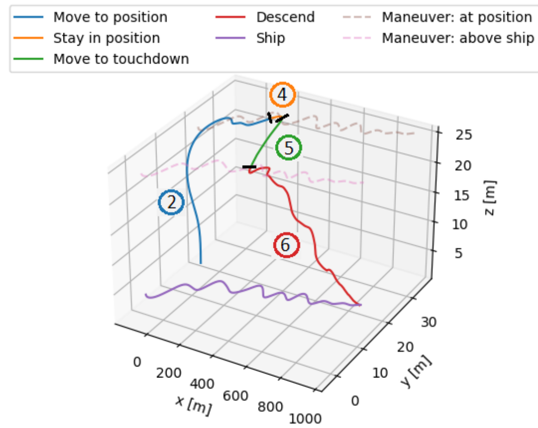

Figure 3: Segmentation result of an UAV 45-Degree ship deck landing maneuver specified in Figure 1.

5 CONCLUSION

We presented our tool for the segmentation of traces using TBT specifications. The tool's robustness verdict of the segmented root node determines whether the given trace satisfied its TBT specification. Additional robustness verdicts of other nodes then further assist in identifying the most challenging parts of the executed behavior. Furthermore, examining alternative segmentations, beyond the optimal one, offers insights into the execution's robustness. For example, users can scrutinize that minor changes in the parameters tau and rho result in the same leaf nodes being assigned. The tool can also serve for debugging unexpected behaviors. For instance, although the operator executed a 45-Degree maneuver the segmentation assigned a Straight-In maneuver. These features have proven to be useful for understanding and validating executed complex behaviors for the use-cases presented in [4].

REFERENCES

[1] Michele Colledanchise and Petter Ögren. 2018. *Behavior trees in robotics and AI: An introduction.* CRC Press.
[2] Alexandre Donzé and Oded Maler. 2010. Robust satisfaction of temporal logic over real-valued signals. In *International Conference on Formal Modeling and Analysis of Timed Systems.* Springer, 92–106.
[3] Georgios E Fainekos and George J Pappas. 2006. Robustness of temporal logic specifications. In *International Workshop on Formal Approaches to Software Testing.*
[4] Sebastian Schirmer, Jasdeep Singh, Emily Jensen, Johann Dauer, Bernd Finkbeiner, and Sriram Sankaranarayanan. 2024. Temporal Behavior Trees: Segmentation and Robustness *(HSCC '24).*