Temporal Behavior Trees: Robustness and Segmentation

Sebastian Schirmer
DLR German Aerospace Center
Braunschweig, Germany
sebastian.schirmer@dlr.de

Jasdeep Singh
University of Colorado Boulder
Boulder, USA
jasdeep.singh@colorado.edu

Emily Jensen
University of Colorado Boulder
Boulder, USA
emily.jensen@colorado.edu

Johann C. Dauer
DLR German Aerospace Center
Braunschweig, Germany
johann.dauer@dlr.de

Bernd Finkbeiner
CISPA
Saarbrücken, Germany
finkbeiner@cispa.de

Sriram Sankaranarayanan
University of Colorado Boulder
Colorado, USA
srirams@colorado.edu

ABSTRACT

This paper presents *temporal behavior trees* (TBT), a specification formalism inspired by behavior trees that are commonly used to program robotic applications. We then introduce the concept of trace segmentation, wherein given a TBT specification and a trace, we split the trace optimally into sub-traces that are associated with various portions of the TBT specification. Segmentation of a trace then serves to explain precisely how a trace satisfies or violates a specification, and which portions of a specification are actually violated. We introduce the syntax and semantics of TBT and compare their expressiveness in relation to temporal logic. Next, we define robustness semantics for TBT specification with respect to a trace. Rather than a Boolean interpretation, the robustness provides a real-valued numerical outcome that quantifies how close or far away a trace is from satisfying or violating a TBT specification. We show that computing the robustness of a trace also segments it into subtraces.Finally, we provide efficient approximations for computing robustness and segmentation for long traces with guarantees on the result.We demonstrate how segmentations are useful through applications such as understanding how novice users pilot an aerial vehicle through a sequence of waypoints in desktop experiments and the offline monitoring of automated lander for a drone on a ship. Our case studies demonstrate how TBT specification and segmentation can be used to understand and interpret complex behaviors of humans and automation in cyber-physical systems.

KEYWORDS

Cyber-physical system, segmentation, temporal behavior trees, temporal logic, offline analysis

ACM Reference Format:
Sebastian Schirmer, Jasdeep Singh, Emily Jensen, Johann C. Dauer, Bernd Finkbeiner, and Sriram Sankaranarayanan. 2024. Temporal Behavior Trees: Robustness and Segmentation. In *27th ACM International Conference on Hybrid Systems: Computation and Control (HSCC '24), May 14–16, 2024, Hong Kong SAR, China.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3641513.3650180

HSCC '24, May 14–16, 2024, Hong Kong SAR, China
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0522-9/24/05
https://doi.org/10.1145/3641513.3650180

1 INTRODUCTION

Behavior trees are increasingly popular in robotic applications. They were originally used in computer games to animate complex sequences of actions for characters. A behavior tree is a programmatic specification of a plan or sequence of possible actions by an autonomous agent. It includes operators that enable us to specify a sequence of sub-plans executed one after the other; falling back to a different sub-plan if the current sub-plan encounters an unexpected failure; repeating a sub-plan multiple times; and conducting many sub-plans in parallel until the number of the sub-plans that have succeeded exceeds a specified lower limit [9]. Figure 1 depicts such a behavior tree for an autonomous landing of an unmanned aerial vehicle (UAV) on a ship deck. It specifies a sequence of operations: (1) move the UAV into position relative to the ship; (2) maintain this position for sometime; (3) move above the touchdown point; and (4) descend onto the ship.

In this paper, we use behavior tree operators in a specification language. Our goal is to describe an acceptable sequence of states/actions in a manner identical to popular temporal logics such as linear temporal logic (LTL), metric temporal logic (MTL), and signal temporal logic (STL). Temporal logics have been widely adopted as a specification language for cyber-physical systems (CPS) and robotics. It forms the backbone for specifying desired behaviors in synthesizing controllers [8, 19, 25, 26, 32]. Nevertheless, the difficulty of writing complex specifications in temporal logic is well-known [18]. Our formalism called *temporal behavior trees* (TBTs)

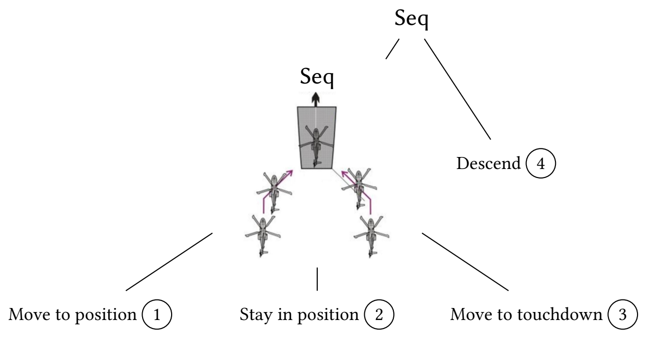


Figure 1: A behavior tree that implements a maneuver for landing on a ship deck. It uses two *Sequence* operators to execute its children from left to right.

use the common behavior tree operators from the literature (sequence, fallback, parallel, kleene, and timeout) with the leaf nodes of the behavior trees decorated by temporal logic formulas, which are given finite trace semantics. We demonstrate that the resulting formalism is strictly more powerful than standard temporal logic with Next, Globally, Finally, and Until operators.

Next, we present the robustness semantics for TBTs in line with the robustness semantics for standard temporal logics introduced by Fainekos and Pappas [13], Donze and Maler [12] and Rizk et al [33]. We interpret a trace under a behavior tree as yielding a real-valued robustness wherein a non-negative value of robustness indicates satisfaction of the trace by the specification while a violation corresponds to a negative robustness. We also show that robustness leads to a segmentation of the trace into subtraces wherein one associates the subtraces with corresponding subtrees of the TBT. Figure 2 shows a segmentation for the maneuver depicted in Figure 1. Assume that the robustness of segment (1) is positive, whereas that for segment (2) is negative. Therefore, we conclude that the UAV moved to the target position as required, but then failed to stay in position. Such a segmentation is very useful in our analysis of the trace: for a trace that violates a specification, it shows specific portions of the trace violated which sub-parts of the overall specification, potentially leading us to diagnostic explanations. It also allows us to rank these failures in terms of the robustness from worst to best. Similarly, for a satisfying trace, we can examine which parts of the trace came closest to violating the corresponding parts of the specifications. Although the exact algorithm for computing robustness and segmentation is at most cubic time in the size of the trace and linear in the size of the tree, we find that in practice this is forbiddingly expensive for long traces with hundreds or thousands of samples per second. We present two systematic approaches to speeding up the computations by orders of magnitude: (a) sub-sampling of the trace using a carefully calculated stride length that preserves the Boolean semantics but approximates the robustness; and (b) a lazy evaluation scheme to rapidly compute an approximate answer.

We show that segmentation is very useful in CPS applications through two empirical case studies based on data from realistic applications. In one study, we examine a detailed TBT specification of a UAV landing on a ship through one of four possible maneuvers, each involving multiple stages to ensure a safe and robust landing even in the presence of disturbances. We also present the use of TBTs and segmentation to analyze human operator performance in flying a UAV inside a desktop-based simulation environment. By specifying the task as a TBT, we demonstrate how segmentation reveals surprising patterns in the data that were not apparent through a whole trace analysis.

1.1 Related Work

Behavior trees (BT) first were invented to enable modular AI in computer games [23], but have become increasingly popular in robotics [1, 15, 20, 22, 35, 38] and beyond [21, 24, 29, 40]. There are several properties that make them attractive including composability, reactivity, and human-readability. The purpose of behavior trees is to specify complex behaviors and execute them. Temporal behavior trees, introduced here, are complementary and can be retrofitted to BT. Their purpose is to specify successful or failed behavior properties that can be used to segment or monitor a trace.

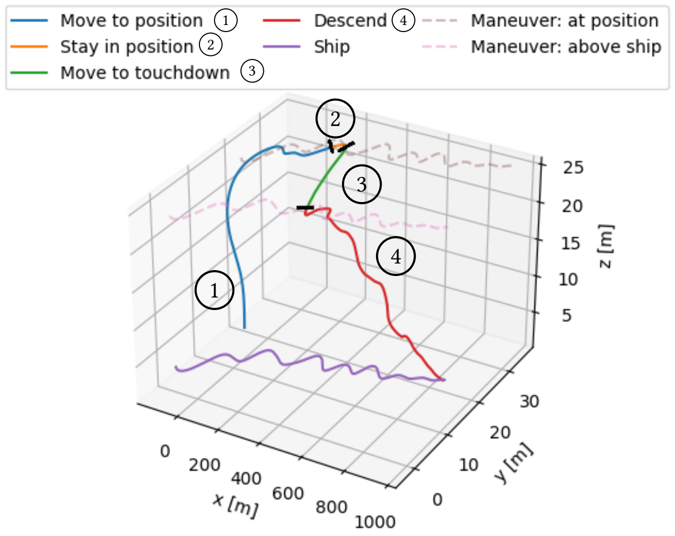


Figure 2: Segmentation result of a UAV 45-Degree ship deck landing maneuver. (1) The UAV starts with moving in position, then (2) it stays for the required amount of time there, before (3) moving above the touchdown, and (4) finally descending. The trajectory of the ship is the purple curve at the bottom with $z = 0$. Dotted lines represents best positions relative to the ship, i.e., left diagonally behind the ship and above touchdown. The various segments are shown in different colors with numbers alongside. These are automatically computed by the algorithm described in this paper.

TBTs defined in this paper have semantics over finite traces. Classically, however, temporal logics have their semantics defined over infinite traces. When it comes to finite traces, the main concern is how to handle *open* obligations at the end of the trace. One approach is to provide a multi-valued semantics such as the one based on so-called *good*, *bad*, and *ugly* prefixes by Bauer et al [3]. Our approach is more closely related to the semantics provided by De Giacomo and Vardi [10]. In fact, the regular language operators presented in that paper are identical to some of the operators used in TBT. In this paper, we additionally consider robustness semantics and define and solve the segmentation problem.

The notion of robustness of a trace with respect to temporal logic specification for *infinite length* trace has its origins independently in the work of Fainekos and Pappas; Donze and Maler; and Rizk et al [12, 13, 33]. The problem of defining and computing robustness of finite-length traces has been studied as well. Deshmukh et al [11] provide a robust interval semantics that maps a finite length trace (seen as a prefix of an infinite trace) and an STL formula to an interval over robustness (l, v) such that for any suffix of the trace, l is the greatest lower bound and v is the lowest upper bound in respect to its robustness value. Our robust semantics of TBT is real-valued, but works over finite traces and introduces robustness for operators that make TBT more expressive when compared to standard temporal logics. The sequence operator for TBTs is similar to the ";" operator in regular linear temporal logic [27] and the "chop operator" in interval temporal logic (ITL) [7]. These ideas go back to the work of Halpern, Manna, and Moszkowski [16] and later further developed by many others, including Harel & Peleg [17] and Rosner & Pnueli [34]. The tool Tempura [1] by Moszkowski and others is an interpreter for executable ITL and can be used similar to a programming language whereas we retrofit temporal logics to executable behaviors trees [30]. Therefore, this paper goes in the reverse direction: we turn a framework used for specifying plans into a temporal logic for specifying properties.

Trace segmentation is closely related to timed pattern matching [39]. However, while timed pattern matching identifies all segments that *satisfy* the pattern criteria, segmentation finds the optimal trace assignment, which may encompass segments that *violate* its specification. The pattern matching algorithm, as outlined in [39], uses a bottom-up zone construction. In contrast, our segmentation algorithm utilizes dynamic programming in a top-down fashion.

2 TEMPORAL BEHAVIOR TREES

In this section, we recap the definition of signal temporal logic (STL), the notion of robustness of a formula to a trace. Next, we define temporal behavior trees (TBTs) by providing their semantics. We show that temporal behavior trees are strictly more expressive than STL thanks to the "sequence" and Kleene operators [41].

A trace is a finite sequence of observable states of a system: $\sigma(1), \ldots, \sigma(N)$, wherein $\sigma(i)$ denotes the state at time $i \in [1, N]$. We write the length of the trace as $|\sigma|$. A trace with $|\sigma| = 0$ is the *empty trace*. Given trace σ, we denote $\sigma[l : u]$ as a slice of the trace from "offsets" l to u (inclusive), as follows:

$$\sigma[l : u] = \begin{cases} \sigma(l+1), \cdots, \sigma(\min(u+1, N)), & \text{if } l \leq u \text{ and } l < N \\ \text{empty trace} & \text{otherwise} \end{cases}$$

For convenience, let $\sigma[l :] = \sigma[l : N-1]$ and $\sigma[: u] = \sigma[0 : u]$.

Example 1. *Note that offsets begin with* 0 *and end at* $|\sigma| - 1$. *For instance,* $\sigma[0 : 3] = \sigma(1), \ldots, \sigma(4)$ *and* $\sigma[5 :] = \sigma(6), \ldots, \sigma(N)$.

Note that many real-time logics treat time as a continuous variable and each state has a corresponding time stamp. Here, we will simply assume that the trace states occur at regular time intervals and thus $\sigma(i)$ is associated with some time $t = i\tau$ for a time period $\tau > 0$. The precise value of τ is not required.

Example 2. *The state* $\sigma(i) : (x_{uav}, y_{uav}, z_{uav}, x_s, y_s, z_s, h_s)$ *for* $1 \leq i \leq N$ *describes the positions of a UAV and a ship, wherein the subscript* s *denotes the state of the ship,* uav *to those of the UAV, and* h_s *is the heading angle of the ship.*

An STL formula extends propositional logic by temporal operators that access future trace events. Let us define a set of *atomic propositions* AP $= \{p_1, \ldots, p_m\}$ where (for the sake of convenience), each p_i is associated with a function f_i that maps states to a real number such that $\sigma \models p_i$ for a state σ iff $f_i(\sigma(1)) \geq 0$.

Definition 1. *STL formulas over AP have the following syntax:*

$$\begin{array}{lll} \varphi & := AP & \longleftarrow \textit{Atomic propositions} \\ & \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi & \longleftarrow \textit{Boolean combinations} \\ & \mid \Diamond_{[l,u]}(\varphi) & \longleftarrow \textit{Eventually with interval} \\ & \mid \Box_{[l,u]}(\varphi) & \longleftarrow \textit{Globally over interval} \\ & \mid \varphi\ \mathcal{U}_{[l,u]}\ \varphi & \longleftarrow \textit{Until with interval} \end{array}$$

Note that some of the formulas are indexed by an interval $[l, u]$ *wherein* $0 \leq l \leq u \leq \infty$. *We write* $\Diamond(\varphi)$ *as a shorthand for* $\Diamond_{[0,\infty)}(\varphi)$, *and similar conventions apply for* $G(\varphi)$ *and* $\varphi_1 \mathcal{U} \varphi_2$. *The "next-state" operator* $\bigcirc(\varphi)$ *is syntactic sugar for* $\Diamond_{[1,1]}(\varphi)$.

In general, STL formulas are defined over infinite traces, but in this paper, we will work with finite traces, which are slices of the form $\sigma[i : j]$ over a given original trace σ. There have been many approaches to provide finite trace semantics for temporal logics [4, 14]. They differ on how they treat the truth of a formula at the end of the trace. We will follow a simple approach along the lines of De Giacomo and Vardi [10], suggested by the standard translation of temporal logics into first-order logic [5]. The satisfaction of an STL formula φ by a finite trace σ is denoted $\sigma \models \varphi$. If φ is an atomic proposition p_i then we have

$$\sigma \models p_i \text{ iff } \begin{cases} f_i(\sigma(1)) \geq 0, & \text{if} |\sigma| > 0 \\ \textit{false}, & \text{if } \sigma \text{ empty.} \end{cases}$$

If $\varphi : \Diamond_{[l,u]}\varphi_1$ then $\sigma \models \Diamond_{[l,u]}\varphi_1$ iff $\exists\, i \in [l, u],\ \sigma[i :] \models \varphi_1$. Recall that $\sigma[i :]$ is the empty trace if $i \geq |\sigma|$. Likewise, for $\varphi : \Box_{[l,u]}\varphi_1$, we have $\sigma \models \Box_{[l,u]}\varphi_1$ iff $\forall\, i \in [l, u],\ \sigma[i :] \models \varphi_1$. Finally, for $\varphi : \varphi_1 \mathcal{U}_{[l,u]} \varphi_2$, we have $\sigma \models \varphi_1 \mathcal{U}_{[l,u]} \varphi_2$ iff

$$\exists i \in [l, u],\ (\forall j \in [0, i-1],\ \sigma[j :] \models \varphi_1) \ \wedge\ \sigma[i :] \models \varphi_2\,.$$

Note that, unlike the semantics for infinite traces, the semantics of STL for finite traces can be non-intuitive especially when the truth of formulas involves reasoning beyond the end of the trace. For instance, the formula $\bigcirc p$ is always false for an empty trace, whereas, $\neg \bigcirc \neg p$ is always true. $\Diamond\varphi$ in a finite trace semantics specifies that φ becomes true before the trace ends. $\Box\varphi$ specifies that φ remains true until the trace ends.

Example 3. *Recalling the definition of a state from Ex. 2, the STL formula* $\Diamond\Box_{[0,5]}\, p_{behind}$ *represents that the position of the UAV should eventually be 20 meters behind the ship, and remain in that position for* 5 *time steps. The function* f_{behind} *allows two meters deviation from a point that is* 20 *meters behind the ship based on its current heading angle:* $f_{behind} = 2 - \sqrt{\begin{array}{c}((x_s + 20 \cdot \cos(180 + h_s)) - x_{uav})^2 + \\ ((y_s + 20 \cdot \sin(180 + h_s)) - y_{uav})^2 + ((z_s + 20) - z_{uav})^2\end{array}}$.

We define the robustness of an STL formula on a trace as a numerical value that we provide to a trace with respect to an STL formula. Non-negative values of robustness denote that a trace satisfies a property whereas negative values indicate the reverse. Finally, the magnitude of the robustness provides us with a measure of how "far away" a trace that satisfies the formula is from violating it, or vice-versa [12, 13]. Robustness measures are useful for a variety of applications, including run-time monitoring and falsification [2].

[1]http://www.antonio-cau.co.uk/ITL/itlhomepagesu14.html

Definition 2 (STL Robust Semantics). *The robustness of an STL formula φ over a trace σ, denoted $\rho(\varphi, \sigma)$, is defined as follows:*

- $\rho(p_i,\ \sigma) = \begin{cases} -\infty, & \text{if trace } \sigma \text{ is empty} \\ f_i(\sigma(1)), & \text{otherwise} \end{cases}$
- $\rho(\neg\varphi,\ \sigma) = -\rho(\varphi, \sigma)$,
- $\rho(\varphi_1 \wedge \varphi_2,\ \sigma) = \min(\rho(\varphi_1, \sigma), \rho(\varphi_2, \sigma))$,
- $\rho(\varphi_1 \vee \varphi_2,\ \sigma) = \max(\rho(\varphi_1, \sigma), \rho(\varphi_2, \sigma))$,
- $\rho(\Diamond_{[l,u]}(\varphi),\ \sigma) = \max_{i\in[l,u]} \rho(\varphi, \sigma[i:])$,
- $\rho(\Box_{[l,u]}(\varphi),\ \sigma) = \min_{i\in[l,u]} \rho(\varphi, \sigma[i:])$,
- $\rho(\varphi_1\ \mathcal{U}_{[l,u]}\ \varphi_2,\ \sigma) = \max_{i\in[l,u]} \min \begin{pmatrix} \rho(\varphi_2, \sigma[i:]), \\ \min_{j\in[0,i-1]} \rho(\varphi_1, \sigma[j:]) \end{pmatrix}$

Theorem 1. *For trace σ and STL formula φ, $\sigma \models \varphi$ iff $\rho(\varphi, \sigma) \geq 0$.*

Proof. We prove the theorem by structural induction on φ. The full proof is given in Appendix A. □

2.1 Syntax and Semantics

We will now define the notion of a temporal behavior tree (TBT), a formalism that borrows operators from behavior trees. A TBT uses operators known from behavior trees that control the order of events and has temporal properties in their leaf nodes.

Definition 3 (Syntax of Temporal Behavior Trees). *Let φ be an STL formula as described by the grammar in Def. 1. We construct a temporal behavior tree using the following syntax:*

$$\begin{array}{lll} \mathcal{T} & := \mathsf{Leaf}(\varphi), & \\ & \mid \mathsf{Fback}([\mathcal{T}, \ldots, \mathcal{T}]), & \\ & \mid \mathsf{Par}_M([\mathcal{T}, \ldots, \mathcal{T}]), & M \in \mathbb{N} \\ & \mid \mathsf{Seq}([\mathcal{T}, \cdots, \mathcal{T}]), & \\ & \mid \mathsf{Tout}_t(\mathcal{T}), & t \in \mathbb{N} \\ & \mid \bigstar_n(\mathcal{T}), & n \in \mathbb{N} \cup \{\infty\} \end{array}$$

Informally, an STL formula at a leaf node specifies that the trace must satisfy the formula. Likewise, $\mathsf{Fback}([\mathcal{T}_1, \ldots, \mathcal{T}_k])$ mimics the semantics of a "fallback" node in a behavior tree: at least one of the subtrees $\mathcal{T}_1, \ldots, \mathcal{T}_k$ must eventually be satisfied by the trace. $\mathsf{Par}_M([\mathcal{T}_1, \ldots, \mathcal{T}_k])$ denotes a parallel operator that specifies that at least M distinct subtrees must be satisfied simultaneously by the trace σ. $\mathsf{Seq}([\mathcal{T}_1, \cdots, \mathcal{T}_k])$ is a sequential node that denotes that σ must be partitioned into k parts (some of which may be empty) $\sigma_1; \sigma_2; \sigma_3; \cdots; \sigma_k$ such that $\sigma_i \models \mathcal{T}_i$. $\mathsf{Tout}_t(\mathcal{T})$ is a timeout node denoting the subtree $\mathcal{T}$ must be satisfied by a prefix of the trace of size t. $\bigstar_n(\mathcal{T})$ is a repeat operator node denoting σ must be partitioned into $k \leq n$ parts $\sigma_1; \sigma_2; \sigma_3; \cdots; \sigma_k$ such that $\sigma_i \models \mathcal{T}$.

The presence of the Seq and $\bigstar$ operators makes temporal behavior trees strictly more expressive than STL (see Theorem 3). Formally, the satisfaction of a TBT specification $\mathcal{T}$ by a trace σ, denoted $\sigma \models \mathcal{T}$ is as follows:

- If $\mathcal{T} : \mathsf{Leaf}(\varphi)$ then $\sigma \models \mathcal{T}$ iff $\sigma \models \varphi$, wherein the semantics of temporal logic have been defined earlier.
- If $\mathcal{T} : \mathsf{Fback}([\mathcal{T}_1, \ldots, \mathcal{T}_k])$ then there exists a $j \in \{1, \ldots, k\}$ and an $i \in [0, |\sigma| - 1]$ such that $\sigma[i:] \models \mathcal{T}_j$. I.e., at least one of the children is *eventually* satisfied by σ.
- $\sigma \models \mathsf{Par}_M([\mathcal{T}_1, \ldots, \mathcal{T}_k])$ iff there exists M distinct indices $i_1, \ldots, i_M \in \{1, \ldots, k\}$ such that $\sigma \models \mathcal{T}_{i_1}$, $\sigma \models \mathcal{T}_{i_2}, \cdots$, and $\sigma \models \mathcal{T}_{i_M}$.
- $\sigma \models \mathsf{Seq}([\mathcal{T}_1, \ldots, \mathcal{T}_n])$ iff there exists distinct indices $i_1 \leq i_2 \leq \ldots \leq i_{n-1}$ such that $\sigma[:i_1] \models \mathcal{T}_1$, $\sigma[i_1 + 1 : i_2] \models \mathcal{T}_2$, $\cdots, \sigma[i_{n-1} + 1 :] \models \mathcal{T}_n$.
- $\sigma \models \mathsf{Tout}_t(\mathcal{T})$ iff $\sigma[: t - 1] \models \mathcal{T}$ if $t \leq |\sigma|$ or $\sigma \models \mathcal{T}$ if $t > |\sigma|$.
- $\sigma \models \bigstar_n(\mathcal{T})$ iff $\exists\, k < n$ indices $i_1 \leq i_2 \leq \ldots \leq i_k$ such that $\sigma[:i_1] \models \mathcal{T}$, $\sigma[i_1 + 1 : i_2] \models \mathcal{T}, \cdots, \sigma[i_k + 1 :] \models \mathcal{T}$.

We now provide some simple examples of TBTs to show how the modularity of TBT helps in practice.

Example 4. *Let $\mathcal{T}$ be a TBT that adds formulas to the leaf nodes of the BT depicted in Figure 1. We can add requirements "on top of" a tree, e.g., $\mathsf{Par}_2([\mathsf{Leaf}(\Box\,\neg obstacle), \mathcal{T}]$ to specify that obstacles are avoided during landing or $\mathsf{Fback}([\mathsf{Tout}_t(\mathcal{T}), \mathsf{Leaf}(contingency)])$ to state that we need to land within t time units or otherwise a contingency needs to be activated. Further, we can add new nodes within $\mathcal{T}$, e.g., we can replace the leaf node 4, referred to as L, with $\mathsf{Seq}([\mathsf{Leaf}(\Box_{[0,t]} above(tp)), L])$ to make sure that we also remain above the touchdown point for a period of time t before descending.*

Let us denote $max_M([i_1, \ldots, i_n])$ as the function that outputs the M^{th} largest number in the list $[i_1, \ldots, i_n]$ if $M \leq n$, and $-\infty$ if $M > n$. Also, for simplicity, in the following we rewrite $\mathsf{Seq}([\mathcal{T}_1, \ldots, \mathcal{T}_k])$ as a sequence $\mathsf{Seq}([\mathcal{T}_1, (\mathsf{Seq}([\mathcal{T}_2, \cdots, \mathsf{Seq}([\mathcal{T}_{k-1}, \mathcal{T}_k]))))$ for $k \geq 2$. Note that $\mathsf{Seq}([\mathcal{T}])$ is the same as $\mathcal{T}$.

Definition 4 (Robustness Semantics of TBT). *The robustness of a temporal behavior tree $\mathcal{T}$ on a finite execution trace σ, denoted $\rho(\mathcal{T}, \sigma)$, is defined as shown in Figure 3. As in the case of STL formulas (Def. 2), it evaluates a given TBT for a trace into a real number, which denotes the "distance to satisfaction" of the specification by the trace.*

We now prove that the robustness ρ for a TBT corresponds to the notion of satisfaction defined above in the following manner.

Theorem 2. *For any trace σ and TBT $\mathcal{T}$, $\sigma \models \mathcal{T}$ iff $\rho(\mathcal{T}, \sigma) \geq 0$.*

Proof. The full proof of the theorem by structural induction on $\mathcal{T}$ is shown in Appendix B. □

STL and TBT are not equivalent in terms of expressiveness. Whereas any property in finite trace STL can be trivially expressed as a TBT with a single leaf node, the converse does not hold.

Theorem 3. *There exists properties of finite traces specified using TBT that cannot be written as a finite trace STL formula.*

Proof. It is well-known that temporal logics based on the $\Diamond, \Box, \mathcal{U}$ and $\bigcirc$ temporal operators cannot express the property "p is *true* in all odd indexed states of a sequence", but TBT can. Pierre Wolper [41] proves this for LTL over infinite traces. Wolper's proof carries over to the version of STL used in this paper. Furthermore, the STL formulas subscripted by finite intervals such as $\Box_{[l,u]}\varphi$ can be written systematically using finitely many nestings of the next operator $\bigcirc$ and the unbounded $\Box$ operator. Similar considerations apply to formulas of the form $\Diamond_{[l,u]}(\varphi)$ and $\varphi_1 \mathcal{U}_{[l,u]} \varphi_2$. Further, the proof extends to finite trace semantics as it only requires a finite prefix p^i that is sufficiently large, i.e., number next operators $l < i$.

$$\rho(\mathsf{Leaf}(\varphi),\ \sigma) = \rho(\varphi, \sigma)$$
$$\rho(\mathsf{Tout}_t(\mathcal{T}), \sigma) = \rho(\mathcal{T}, \sigma[:\min(|\sigma|-1, t-1)])$$
$$\rho(\mathsf{Seq}([\mathcal{T}_1, \mathcal{T}_2]), \sigma) = \max_{u\in[0,|\sigma|-1]} \min\begin{pmatrix} \rho(\mathcal{T}_1, \sigma[:u]), \\ \rho(\mathcal{T}_2, \sigma[u+1:]) \end{pmatrix}$$
$$\rho(\mathsf{Fback}([\mathcal{T}_1, \dots, \mathcal{T}_k])),\ \sigma) = \max_{j\in[1,k]} \max_{i\in[0,|\sigma|-1]} \rho(\mathcal{T}_j, \sigma[i:])$$
$$\rho(\mathsf{Par}_M([\mathcal{T}_1, \dots, \mathcal{T}_k]), \sigma) = max_M\left(\ \rho(\mathcal{T}_1, \sigma),\ \cdots,\ \rho(\mathcal{T}_k, \sigma)\ \right)$$
$$\rho(\bigstar_n(\mathcal{T}), \sigma) = \begin{cases} \rho(\mathsf{Seq}([\mathcal{T}, \bigstar_{n-1}(\mathcal{T})]), \sigma), & \text{if } |\sigma| > 0 \wedge n > 0 \\ \infty, & \text{otherwise.} \end{cases}$$

Figure 3: Definition of robustness semantics for temporal behavior trees.

However, the *even* property can be expressed as

$$\bigstar_\infty(\mathsf{Seq}([\neg \bigcirc\, \mathit{true} \wedge p, \neg \bigcirc\, \mathit{true}]))$$

Note that $\neg \bigcirc\, \mathit{true}$ enforces Seq to chop the trace after reading one position, as discussed in Sec. 2.1. The $\bigstar$-operator repeats this until reaching the end of the trace. □

2.2 Computing Robustness

We will now provide a dynamic programming approach to compute the robustness efficiently. This will be used to define the notion of trace segmentation in the next section. For convenience, let σ be the given original trace.

Definition 5 (Dynamic Programming for TBT Robustness). *The robustness of a temporal behavior tree $\mathcal{T}$ on a finite execution trace σ can be implemented in a dynamic programming fashion by computing $\rho_\sigma(\mathcal{T}, i, j) = \rho(\mathcal{T}, \sigma[i:j])$, as shown in Figure 4.*

The computed robustness values that are stored in a Memo-Table T can be accessed by $\mathrm{T}(\mathcal{T}, i, j)$ where $\mathcal{T}$ is a (sub-)TBT, i is the beginning of the segment, and j its end. The size of the table is $O(N^2|\mathcal{T}|)$ given a trace of size $|\sigma| = N$. However, the time taken to fill out each entry in the memo table can be $O(N)$ in the worst case (see the definition of the Seq node). It also depends on the time taken to compute the robustness of the temporal logic formulas at the leaf, which we assume will take time $O(N^2|\varphi|)$, for a leaf node $\mathsf{Leaf}(\varphi)$, wherein $|\varphi|$ denotes the size of the formula. Assuming that $|\mathcal{T}|$ includes the size of the temporal logic formulas at the leaves, as well, we can bound the execution time as $O(N^3|\mathcal{T}|)$.

Lemma 1. *For any trace σ and TBT $\mathcal{T}$, the value of $\rho_\sigma(\mathcal{T}, 0, |\sigma|-1)$ from Definition 5 is the same as $\rho(\mathcal{T}, \sigma)$ from Definition 4.*

$$\rho_\sigma(\mathsf{Leaf}(\varphi), i, j) = \rho(\varphi, \sigma[i:j]) \text{ (Cf. Definition 2)},$$
$$\rho_\sigma(\mathsf{Fback}([\mathcal{T}_1, \dots, \mathcal{T}_k]), i, j) = \max_{l\in[1,k]} \max_{i'\in[i,j]} (\rho_\sigma(\mathcal{T}_l, i', j)),$$
$$\rho_\sigma(\mathsf{Par}_M([\mathcal{T}_1, \dots, \mathcal{T}_k]), i, j) = \max_M(\rho_\sigma(\mathcal{T}_1, i, j), \dots, \rho_\sigma(\mathcal{T}_k, i, j)),$$
$$\rho_\sigma(\mathsf{Seq}([\mathcal{T}_1, \mathcal{T}_2]), i, j) = \max_{u\in[i,j]} \min(\rho_\sigma(\mathcal{T}_1, i, u), \rho_\sigma(\mathcal{T}_2, u+1, j))$$
$$\rho_\sigma(\mathsf{Tout}_t(\mathcal{T}), i, j) = \rho_\sigma(\mathcal{T}, i, \min(j, i+t-1)),$$
$$\rho_\sigma(\bigstar_n(\mathcal{T}), i, j) = \begin{cases} \rho_\sigma(\mathsf{Seq}([\mathcal{T}, \bigstar_{n-1}(\mathcal{T})]), i, j), & \text{if } n > 0 \text{ and } i \le j \\ \infty, & \text{if } n = 0 \text{ or } i > j \end{cases}$$

Figure 4: Robust semantics of temporal behavior tree expressed in a form suitable for dynamic programming.

Proof. We prove by induction on the structure of the tree, that for any two indices i, j, we have $\rho_\sigma(\mathcal{T}, i, j) = \rho(\mathcal{T}, \sigma[i:j])$. Base case is for leaf formulas and formulas of the form $\bigstar_n(\mathcal{T})$ for $n = 0$ or $i > j$. In all cases, a comparison of the cases in Figures 3 and 4 shows that they yield the same values. The proof compares each node type to show that if ρ_σ and ρ agree on the children of a node, they also agree on the node itself. Comparing Figures 3 and 4 shows that they yield the same values. □

3 SEGMENTING TRACES

In this section, we define the problem of segmenting traces with respect to a TBT specification. We show the connection between segmentation and the robustness semantics as in Definition 4. The dynamic programming formulation that computes robustness also computes the segmentation of a trace. We briefly describe how the segmentation can be useful, with further demonstrations of usefulness provided in Section 4.

We begin by defining the segmentation of a trace with respect to a TBT $\mathcal{T}$. Informally, given a TBT $\mathcal{T}$, the segmentation of a trace σ splits it into multiple subtraces of the form $\sigma[i:j]$ such that (a) every subtree of $\mathcal{T}$ is associated with a subtrace; and (b) the satisfaction and robustness of the specification $\mathcal{T}$ by the trace σ can be linked to the satisfaction and robustness of each of the subtraces $\sigma[i:j]$ associated with the corresponding subtree $\hat{\mathcal{T}}$. We recall the definition of robustness $\rho(\mathcal{T}, \sigma)$ from Definition 5 and Figure 4.

Definition 6 (Segmentation of a TBT). *The segmentation of a trace σ with respect to a TBT $\mathcal{T}$ is a graph $G = (V, E)$ whose vertex set V consists of triples of the form*

$$V = \{(\hat{\mathcal{T}}, i, j) \mid \hat{\mathcal{T}} \text{ is a subtree of } \mathcal{T},\ 0 \le i, j \le |\sigma| - 1\},$$

and edges $E \subseteq V \times V$ such that the following conditions hold:

1. *$(\mathcal{T}, 0, |\sigma|-1) \in V$ corresponding to the entire tree $\mathcal{T}$ and the entire trace from indices 1 to $|\sigma|$.*
2. *If a node v is of the form $(\mathsf{Fback}([\mathcal{T}_1, \dots, \mathcal{T}_k]), i, j) \in V$, and $i \le j$ then there is precisely one subtree index $l \in [1, k]$ and a single trace index $i' \in [i, j]$ such that the edge $v \longrightarrow (\mathcal{T}_l, i', j) \in E$.*
3. *If a node v of the form $(\mathsf{Seq}([\mathcal{T}_1, \mathcal{T}_2]), i, j) \in V$, there exists a unique index u such that $(\mathcal{T}_1, i, u) \in V$, $(\mathcal{T}_2, u+1, j) \in V$ and the edges $v \longrightarrow (\mathcal{T}_1, i, u)$ and $v \longrightarrow (\mathcal{T}_2, u+1, j)$ belong to E.*
4. *If a node v of the form $(\mathsf{Par}_M([\mathcal{T}_1, \dots, \mathcal{T}_k]), i, j) \in V$, we have precisely M distinct indices $l_1, \dots, l_M \in [1, k]$ such that the set $S = \{(\mathcal{T}_{l_1}, i, j), \cdots, (\mathcal{T}_{l_M}, i, j)\} \subseteq V$ and edges from v to each of the nodes in S belong to E.*

(5) *If a node v of the form $(\mathsf{Tout}_t(\mathcal{T}_1), i, j) \in V$, then the node $v' = (\mathcal{T}_1, i, \min(j, i + t - 1)) \in V$ with an edge from v to v' in E.*
(6) *If v is of the form $(\bigstar_n(\mathcal{T}), i, j) \in S$ and $n \geq 1$ and $i \leq j$, then either (a) the node $v' = (\mathcal{T}, i, j) \in V$ with an edge from v to v'; or (b) there exists indices $u_1 \leq \ldots \leq u_k$ for some $1 \leq k < n$ such that the set of nodes $S = \{(\mathcal{T}, i, u_1), (\mathcal{T}, u_1 + 1, u_2), \ldots, (\mathcal{T}, u_k + 1, j)\} \subseteq V$ have edges in E from v to each node in S.*
(7) *The set of vertices V and edges E are minimal: i.e, no proper subsets of V, E satisfy the conditions stated above.*

Note that the last condition of minimality is important to ensure that we do not add unneeded vertices and edges to a segmentation.

Lemma 2. *Any segmentation of a trace σ and TBT $\mathcal{T}$ is a directed acyclic graph. Furthermore, a node has no outgoing edges if and only if it is of the form (a) $(\mathsf{Leaf}(\varphi), i, j)$, (b) $(\bigstar_0, i, j)$, or (c) $(\bigstar_n, i, j)$ with $i > j$.*

Proof. Each edge $v : (\mathcal{T}, i, j) \to v' : (\mathcal{T}', i', j')$ in the set E from Definition 6 goes from a TBT $\mathcal{T}$ to its subtree $\mathcal{T}'$. Therefore, no cycles can exist in the graph G. The second part also follows from Definition 6. □

Example 5. *Consider a trace σ of length 100 and a specification $\mathcal{T}$ of the form $\mathsf{Fback}([\mathcal{T}_1, \mathcal{T}_2])$ where $\mathcal{T}_1$ is $\mathsf{Seq}([\mathsf{Leaf}(\varphi_1), \mathsf{Leaf}(\varphi_2)])$ and $\mathcal{T}_2$ is $\mathsf{Seq}([\mathsf{Leaf}(\varphi_2), \mathsf{Leaf}(\varphi_1)])$. The TBT expresses the fact that eventually φ_1 must be satisfied followed by φ_2, or the other way around. One possible segmentation (graph) is provided in Fig. 5 (left).*

Consider another TBT of the form $\bigstar_3(\mathsf{Seq}([\mathsf{Leaf}(\varphi_1), \mathsf{Leaf}(\varphi_2)]))$. An example segmentation is shown in Figure 5 (right).

Given a segmentation graph $G = (V, E)$ for a TBT $\mathcal{T}$ and trace σ, it induces a value for the robustness of the specification and trace *given the segmentation.* Formally, we will define $\pi_G(\hat{\mathcal{T}}, i, j)$ for each $(\hat{\mathcal{T}}, i, j) \in V$ as follows:

(1) If $\hat{\mathcal{T}} = \mathsf{Leaf}(\varphi)$ then $\pi_G(\hat{\mathcal{T}}, i, j) = \rho(\varphi, \sigma[i : j])$ using Def. 2.
(2) If $\hat{\mathcal{T}} = \bigstar_0(\mathcal{T})$ or the node is of the form $(\bigstar_n, i, j)$ with $i > j$, then $\pi_G(\hat{\mathcal{T}}, i, j) = \infty$.
(3) Otherwise, $\pi_G(\hat{\mathcal{T}}, i, j) = \min(\{\pi_G(\mathcal{T}_1, i', j') \mid (\hat{\mathcal{T}}, i, j) \to (\mathcal{T}_1, i', j') \in E\})$, i.e., to the minimum robustness of its given existing successors in G.

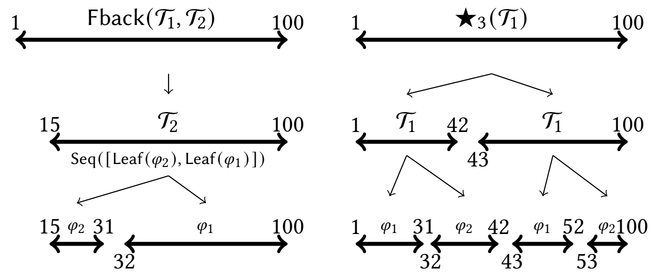


Figure 5: Example trace segmentations for the two properties described in Ex. 5.

Let $\mathsf{segs}(\mathcal{T}, \sigma)$ denote all possible segmentations of the trace σ against the TBT specification $\mathcal{T}$. We say that a segmentation G is optimal iff $\pi_G(\mathcal{T}, 0, |\sigma| - 1) = \max_{G' \in \mathsf{segs}(\mathcal{T}, \sigma)} \pi_{G'}(\mathcal{T}, 0, |\sigma| - 1)$. In other words, the robustness calculated by the segmentation is maximal among all possible segmentations of the trace with respect to the specification. Informally, this means that the trace is segmented in the "best possible light" in an attempt to match it against the specification.

Theorem 4. *Let G be an optimal segmentation of σ w.r.t. $\mathcal{T}$. It follows that $\pi_G(\mathcal{T}, 0, |\sigma| - 1) = \rho(\mathcal{T}, \sigma)$.*

Let us consider an example segmentation for the STL formula provided in Example 3.

Example 6. *Consider once again the STL formula used in Ex. 3. The target position p is defined by the euclidean distance as before. Let $\mathsf{Seq}([\mathsf{Leaf}(\lozenge p), \mathsf{Leaf}(\square_{[0,5]}\ p)])$ be the corresponding TBT for the formula. In this way, a segmentation allows us to identify which child was successful or not. For instance, given a segmentation that contains $(\mathsf{Leaf}(\lozenge p), 0, 99)$ and $(\mathsf{Leaf}(\square_{[0,5]}, 100, 199)$ for a trace σ, where $\rho(\mathsf{Leaf}(\lozenge p), \sigma[: 99]))$ is positive and $\rho(\mathsf{Leaf}(\square_{[0,5]}, \sigma[100 :]))$ is negative, we can conclude that the UAV reached position p but did not hold that position for as long as necessary.*

Computing an Optimal Segmentation. We will now turn to the problem of computing an optimal segmentation given a trace σ and TBT $\mathcal{T}$. As it turns out, an optimal segmentation can be recovered through the dynamic programming table $\mathrm{T}(\hat{\mathcal{T}}, i, j)$ used to compute $\rho_\sigma(\mathcal{T}, 0, |\sigma| - 1)$. We will outline the steps of this computation below. To begin with, we will use a worklist of unprocessed vertices. Whenever the worklist is empty, we have discovered all the nodes and edges of the optimal segmentation G. We initialize the worklist to contain the vertex $(\mathcal{T}, 0, |\sigma| - 1)$. At each iteration, we pop a node $v = (\hat{\mathcal{T}}, i, j)$ from the worklist:

(1) If $\hat{\mathcal{T}} = \mathsf{Leaf}(\varphi)$, $\hat{\mathcal{T}} = \bigstar_0(\mathcal{T}')$ or $\hat{\mathcal{T}} = \bigstar_n(\mathcal{T}')$ with $i > j$, we add no outgoing edges.
(2) If $\hat{\mathcal{T}} = \mathsf{Fback}([\mathcal{T}_1, \ldots, \mathcal{T}_k])$, let
$$(l, i') = \mathrm{argmax}_{(l,i') \in [1,k] \times [i,j]} \mathrm{T}(\mathcal{T}_l, i', j)\,.$$
We add the vertex $v' = (\mathcal{T}_l, i', j)$ to the worklist if it does not exist previously and the edge $v \to v'$.
(3) If $\hat{\mathcal{T}} = \mathsf{Seq}([\mathcal{T}_1, \mathcal{T}_2])$, let $u = \mathrm{argmax}_{u \in [i,j]}(\min(\mathrm{T}(\mathcal{T}_1, i, u), \mathrm{T}(\mathcal{T}_2, u + 1, j)))$. We add the vertices $v_1 = (\mathcal{T}_1, i, u)$, $v_2 = (\mathcal{T}_2, u + 1, j)$ to the worklist (if they did not exist previously) and the edges $v \to v_1, v \to v_2$.
(4) If $\hat{\mathcal{T}} = \mathsf{Par}_M([\mathcal{T}_1, \ldots, \mathcal{T}_n])$, the list $[\mathrm{T}(\mathcal{T}_1, i, j), \ldots, \mathrm{T}(\mathcal{T}_n, i, j)]$ is sorted in descending order, and take the first M entries in the sorted list. We add the vertices $(\mathcal{T}_{i_1}, i, j), \ldots, (\mathcal{T}_{i_M}, i, j)$ corresponding to the first M sorted entries to the worklist (if they did not exist previously) and outgoing edges form v to all these vertices.
(5) If $\hat{\mathcal{T}} = \bigstar_n(\mathcal{T}_1)$ and $n > 0$, $i \leq j$, we take the arguments of the maximum of $\forall k \in [1, n].u_1, \ldots, u_k = \arg$
$$\max_{u_1 \in [i,j], \ldots, u_k \in [u_{k-1}, j]} \min\big(\ \mathrm{T}(\mathcal{T}_1, i, u_1), \ldots,\ \mathrm{T}(\mathcal{T}_1, u_k, j)\ \big).$$
We add the vertices $(\mathcal{T}_1, i, u_1 + 1), \ldots, (\mathcal{T}_1, u_k + 1, j)$ to the worklist (if new) and create edges from v to these vertices.

Let G represent the segmentation thus obtained.

Theorem 5. *The segmentation obtained using the procedure described above is optimal: i.e, $\pi_G(\mathcal{T}, \sigma) = \rho(\mathcal{T}, \sigma)$.*

The proof of the theorem follows from the construction itself since it simply "reads off" a segmentation from the dynamic programming table T.

Dynamic programming also allows us to rapidly explore alternative segmentations. Alternative segmentations allow to go beyond the optimal segmentation and help to better understand the executed behavior, e.g., to identify if there are "other" successful segmentations. We control finding such alternatives using two parameters: τ_t and τ_ρ. $\tau_t \in \mathbb{R}^+$ represents how much the segment boundary needs to differ in respect to already identified segmentations. $\tau_\rho \in \mathbb{R}^+$ represents a lower bound of the robustness of alternatives in respect to given segmentations.

Definition 7 (Alternative Segmentation). *Let σ be a trace and $\mathcal{T}$ be a temporal behavior tree, a segmentation $G' = (V', E')$ is an alternative to a segmentation $G = (V, E)$ iff $\pi_{G'}(\mathcal{T}, \sigma) > \tau_\rho$ and either $V \neq V'$ or there exists a sink vertex $(\mathcal{T}_k, i, j) \in V$ with no sink vertex $(\mathcal{T}'_k, i', j') \in V'$ for which $\mathcal{T}_k = \mathcal{T}'_k$ and $|i-i'|+|j-j'| \leq \tau_{time}$.*

Example 7. *Consider the segmentation provided in Figure 2. As can be seen,* Stay at Position *was assigned to the subtrace of σ, annotated by (2), just before moving towards the ship (3). Using alternative segmentation, we can analyze whether (2) and (3) have an earlier or later assignment that also satisfies the specification, i.e., was it possible to execute the next leaf node earlier or later. To obtain only satisfying alternatives, we choose $\tau_\rho = 0$. We also choose τ_{time} as the duration of staying in position as specified to make sure that our alternative is significantly different to the one obtained before. Note that by choosing $\tau_{time} = |\sigma|$, we are guaranteed to find an alternative that uses a different landing maneuver, if existing.*

Computing an alternative segmentation uses the same worklist algorithm presented above, but imposes the additional constraints defined in Def. 7 while accessing T. We will present the detailed approach in an extended version of this paper.

Subsampling traces for approximating robustness. Traces are often obtained by sampling a continuous signal at regular time intervals. Often, if the sampling is done rapidly compared to how the signal varies, we observe the phenomenon of *stuttering*, wherein the same truth values of atomic propositions repeat over multiple time instances. We now show how the truth/robustness on a carefully subsampled trace relates to the original trace.

Definition 8 (δ-stuttering). *A trace σ is δ-stuttering iff (a) δ divides $|\sigma|$ and (b) for all i such that $i\delta < |\sigma|$, the states $\sigma(i\delta + 1), \ldots, \sigma((i+1)\delta)$ are identical in terms of the atomic proposition truth valuations: $\sigma(i\delta+1) \models p$ iff $\sigma(i\delta + j) \models p$ for $j = \{1, \ldots, \delta\}$.*

We say that a trace σ' is a subsampling of σ with *stride length* $\delta \geq 1$ iff $\sigma'(i+1) = \sigma(i\delta + 1)$ for $i \in \{0, \ldots, \frac{|\sigma|}{\delta} - 1\}$.

Definition 9 (δ-preserving TBT). *A TBT $\mathcal{T}$ is δ-preserving if (a) all occurrences of $\mathcal{U}_{[l,u]}$ in $\mathcal{T}$ has $l = 0$ and u is divisible by δ; (b) all occurrences of $\Diamond_{[l,u]}, \Box_{[l,u]}$ have l, u divisible by δ; and (c) all occurrences of* Tout_t *has t divisible by δ.*

For δ-preserving TBT $\mathcal{T}$, define TBT $\mathcal{T}'$ with all occurrences of Tout_t replaced by $\mathsf{Tout}_{t/\delta}$ and all occurrences of temporal logic operators $\bowtie_{[l,u]}$ wherein $\bowtie \in \{\Diamond, \Box, \mathcal{U}\}$ replaced with $\bowtie_{[l/\delta, u/\delta]}$.

Example 8. *Let TBT $\mathcal{T}$ be* $\mathsf{Seq}([\Box(a \wedge \neg b), \Diamond b])$ *and let σ be a trace $\begin{smallmatrix} a: \\ b: \end{smallmatrix}\binom{true}{false}, \binom{true}{false}, \binom{false}{true}, \binom{false}{true}$, then $\delta = 2$, $\mathcal{T}'$ remains $\mathcal{T}$, and σ' is the sequence $\binom{true}{false}, \binom{false}{true}$, and $\rho_\delta(\mathcal{T}, \sigma)$ is positive.*

Let $\mathcal{T}$ be δ-preserving TBT and σ be a δ-stuttering trace.

Theorem 6. *If $\sigma' \models \mathcal{T}'$ then $\sigma \models \mathcal{T}$.*

Proof. We prove the theorem by induction on the formula. The full proof is given in Appendix C. □

The theorem shows that by evaluating a TBT on a subsampled trace for a δ-stuttering trace σ, if the subsampled trace satisfies the TBT, then so does the original trace. However, violations on a subsampled trace need not necessarily be violations on the original trace. The Seq operator is the reason for the failure of the converse.

Example 9. *Consider a trace σ with atomic proposition p: $\neg p, \neg p, \neg p, p, p, p, \neg p, \neg p, \neg p$, and TBT $\mathcal{T}$:* $\mathsf{Seq}(\mathsf{Leaf}(\Diamond p), \mathsf{Leaf}(\Diamond p))$. *Choosing $\delta = 3$, we obtain σ' : $\neg p, p, \neg p$. The transformed TBT $\mathcal{T}' = \mathcal{T}$. Note that $\sigma \models \mathcal{T}$ whereas $\sigma' \not\models \mathcal{T}'$.*

Also, it is necessary for $\mathcal{U}_{[l,u]}$ occurrences to have $l = 0$. Consider the trace σ : p, p, p, q, q, q and the formula φ : $\Box((p\,\mathcal{U}_{[3,3]}\,q) \vee q)$. Clearly $\sigma \not\models \varphi$. However, we have $\delta = 3$ and thus σ' : p, q with φ' : $\Box((p\,\mathcal{U}_{[1,1]}\,q) \vee q)$. It follows that $\sigma' \models \varphi$.

Note that, for simplicity, we computed the stride length δ based on the fact that $p \in AP$ remains unchanged in every subtrace $\sigma[i\delta : (i+1)\delta - 1]$. This is too stringent in practice. For instance, the formula $\Box(p_1 \vee p_2 \wedge p_3) \vee p_4$ with $AP = \{p_1, p_2, p_3, p_4\}$ will be replaced by AP' that consists of p_4 and q where $f_q = \max(f_{p_1}, \min(f_{p_2}, f_{p_3}))$. For our experiments in Section 4, we implemented this improvement to increase δ but also to reduce the size of the TBT $\mathcal{T}$.

Approximate Robustness Using Lazy Evaluation. Next, we introduce a lazy evaluation for computing the robustness of a TBT given a trace. For the lazy evaluation, we use operators common in functional and dynamic programming that rely on iterators and coroutines [28, 31]. Using these concepts, we define $\min_{lazy}$ as follows:

```
define min_lazy(list_of_exprs):
    min_so_far = ∞
    # initialize lazy evaluation of all expressions
    list_of_gens = [lazy_eval(e) for e in list_of_exprs]
    for g in list_of_gens:
        if has_next(g):
            l = next(g) # take the next value yielded
            if l < 0 and l < min_so_far:
                yield
            min_so_far = min(min_so_far, l)
    return min_so_far
```

We define $\max_{lazy}$ in an analogous manner. The key idea here is that a call to $\min_{lazy}$ will yield as soon as it finds a negative value or a value smaller than what it yielded previously. Many programming language implement these features, e.g, Python [2]. We execute the dynamic programming starting until the overall root expression

[2] https://realpython.com/introduction-to-python-generators/

yields its first value and stops. Even if this value is not the exact robustness, we can relate this value to the exact result obtained upon a full computation.

Definition 10 (Lazy-Evaluation). *Given a TBT $\mathcal{T}$ and a trace σ, we can evaluate the robustness in a lazy manner, denoted as $\rho_{lazy}(\mathcal{T}, \sigma)$, by replacing all instances of* max *by* $\max_{lazy}$ *and* min *by* $\min_{lazy}$ *in* $\rho(\mathcal{T}, \sigma)$. *In addition, we will also use memorization to cache previously evaluated expressions in order to avoid re-evaluation.*

Example 10. *Given the same TBT and trace as in Ex. 8. The lazy evaluation starts by sequencing the trace right after the first position into two segments. While computing the robustness of the left segment using $\Box$ no lazy return happens since there is only one position. When computing the right side using $\Diamond$ the first lazy return occurs right for the second value provided by the iterator. The evaluation returns to the sequence node, where both segments have a positive robustness. Hence, lazy return will directly return the minimum of these segments. If only one of the segments would have been negative, the evaluation would backtrack to the right segment and continues to call the iterator at the position where it returned before. Note that in this case, the computed robustness is optimal. This is not necessarily the case.*

Theorem 7. *Given a TBT $\mathcal{T}$ and a trace σ, $\rho_{lazy}(\mathcal{T}, \sigma) \geq 0$ iff $\rho(\mathcal{T}, \sigma) \geq 0$.*

Proof. The same algorithm as $\rho(\mathcal{T}, \sigma)$ is used and coroutines maintain an internal state that allows them to continue an evaluation of $\max_{lazy}$ and $\min_{lazy}$ when returning to them. $\Box$

4 EMPIRICAL EVALUATION AND CASE-STUDIES

In this section, we will present different case studies that show how the segmentation of TBT provides useful insights. Note that formalizing both use-cases using STL is not possible because our TBT specification uses the "Sequence" operator, which allows to chop a trace – an operation not supported by any STL operator.

4.1 Analysis of Human Behavior

We use segmentation to analyze human operator performance. We obtained data from the study conducted by Byeon et al, wherein thirteen subjects repeatedly attempted to fly a drone in a simulated environment using a joystick setup to control the drone's altitude and attitude while avoiding the obstacle [6]. Each subject attempted the same task of navigating through the waypoints 25 times. The goal of the simulation was to see if the operator through these repeated trials will learn how to navigate the drone to take off and fly through six different waypoints, ending up at a pre-specified position. We used the TBT shown in Figure 6 to specify the overall task. Note that the entire task is a sequence of moves from one waypoint to the next reaching the terminus.

We use segmentation of each of the traces of (x, y, z) trajectories of the human operator to understand how their performance evolved over the trials. Note that segmentation is challenging since human operators often behave in ways we could not predict a priori. Figure 7 shows the segmentations obtained by our approach for two different operators and trials (there are a total of 13×25 such traces). The segmentations are obtained automatically given the

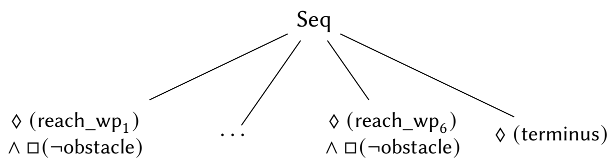

Seq
$\Diamond$ (reach_wp$_1$) $\wedge \Box(\neg$obstacle)
. . .
$\Diamond$ (reach_wp$_6$) $\wedge \Box(\neg$obstacle)
$\Diamond$ (terminus)

Figure 6: Temporal behavior tree that was used to specify the task for each participant in the drone flying task. The task consists of reaching a sequence of 6 waypoints while avoiding two obstacles in turn and ending up at a terminus.

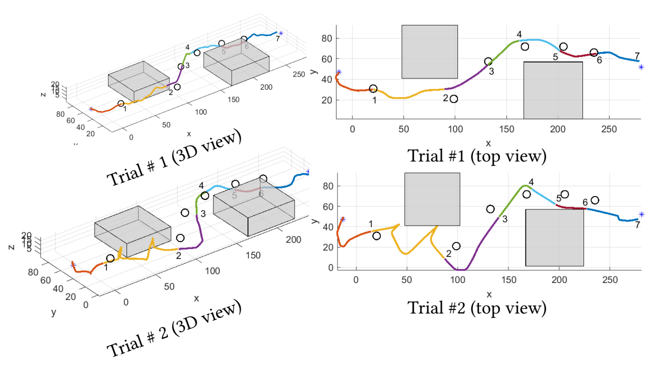


Figure 7: Segmented trajectories of two different simulator flights. Each segment is drawn in a different color and numbers are shown at the ending point of each segment. The 3D plots are shown to the left and the top view is shown to the right. Waypoints are shown as circles and obstacles are shaded in gray.

traces and the specification. They split the entire trajectory into seven parts, ascribing each to a subtree which in this case corresponds to reaching a waypoint. Note from Figure 7 that it is often hard to perform this segmentation manually especially when the specifications are violated (they are violated in both cases due to missed waypoints and collision with the obstacle in Trial #2). The segmentation allows us to analyze how the subjects are learning or failing to learn the performance of the overall task over each trial. For each trial, we collect the robustness of each segment with respect to its corresponding node in the tree.

The plots in Figure 8 reveal consistent trends that were observed across all the subjects: (a) Most subjects could carry out the first phase successfully and the overall spread of robustness is relatively small. (b) The second phase ends up being the most challenging. Very few subjects over few trials could navigate this successfully. We suspect that the presence of the obstacle right next to the waypoint for this phase plays a role in this. (c) Subjects are able to navigate phases 3-6 successfully on average but their performance varies across trials. (d) Subjects are consistently unable to navigate the terminal phase. Our preliminary analysis clearly demonstrates the usefulness of a systematic approach to segmentation. The full analysis of how humans learn to teleoperate successfully over the course of multiple trials will be described in an extended version.

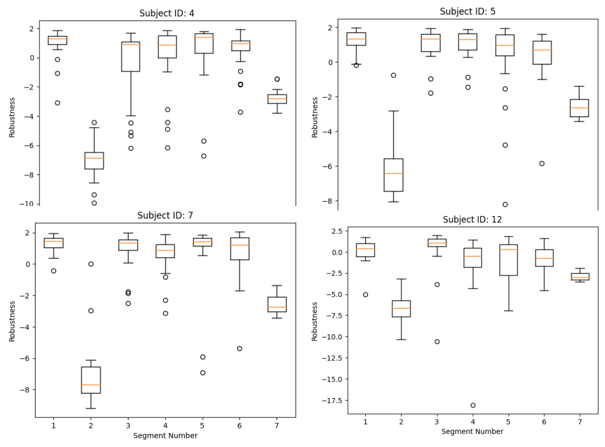


Figure 8: Box plots showing the spread of robustness values corresponding to each of the seven task segments for four different subjects.

4.2 Autonomous Ship Deck Landing

Landing on a ship deck is well-known to be a challenging task, wherein various landing aids and maneuvers need to be carefully selected [36, 37]. The TBT presented in Figure 10 formally specifies four different landing maneuvers: *Straight-in*, *Lateral*, *45-Degree*, and *Oblique*. At the top of the TBT, there is a sequence node that executes its children from left to right. The first child is a fallback node and the second is a leaf node that represents the descent to the touchdown point. Each maneuver is structured as a sequence with different atomic propositions. The maneuvers differ in their starting position and their heading. Whereas, the 45-Degree and the Oblique landing maneuvers specify that the UAV must be diagonally behind the ship, they differ in the relative heading of the UAV to the ship, i.e., aligned with versus oblique to the ship heading, respectively.

We use our TBT on simulation data provided by the authors of [36, 37] for landing with a UAV on a ship deck under wind conditions from the side (WS) and from the front (WF). All experiments were run on a single 16-core machine with a 2.50 GHz 11th Gen Intel(R) Core(TM) i7-11850H processor with 32 GB RAM. The algorithms are implemented as a single-threaded program using Rust [3]. Experimental results are given in Table 1. The first column indicates the expected behavior for a landing. Each of the logfiles have mission times between 115 and 127 seconds and contain between 22,046 and 25,313 entries. The second column reports the stride length δ. Next, *Time* represents the execution time required to find a segmentation where ρ_δ refers to the presented subsampling and $\rho_\delta \circ \rho_{lazy}$ refers to first subsampling and then running a lazy evaluation (Def. 10). Finally, we report on the *Chosen Maneuver* and the *Worst Segment* given the computed segmentation. For the case where only subsampling was used, at least 2,000,000 and up to 500,000,000 evaluations could be resolved by the memo table. The results show that subsampling is very efficient and helps to find segmentations within seconds. Yet, the dependence on the length of the trace becomes clear when we compare the required time required for $\delta = 25$ and $\delta = 200$. Our segmentations show that oblique maneuvers potentially failed. This also caused the lazy evaluation to take as long as ρ_δ because early returns of $\min_{lazy}$ and $\max_{lazy}$ are not possible in this case. Further, we can see that the most challenging part of the maneuver is *Descend*, which makes sense given the disturbances due to wind and waves when landing on the ship. In fact, segmentation of the *Straight-in-WF* logfile potentially failed due to its descend. Note that the best robustness for a *Descend* is one since we allow a tolerable deviation from the target position by one meter, see Figure 10. Results indicate that wind did not play a major role during the maneuvers.

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

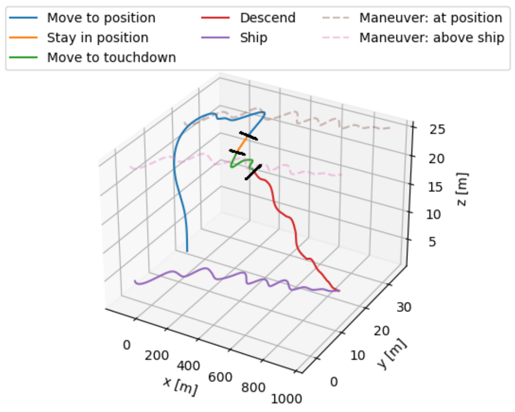
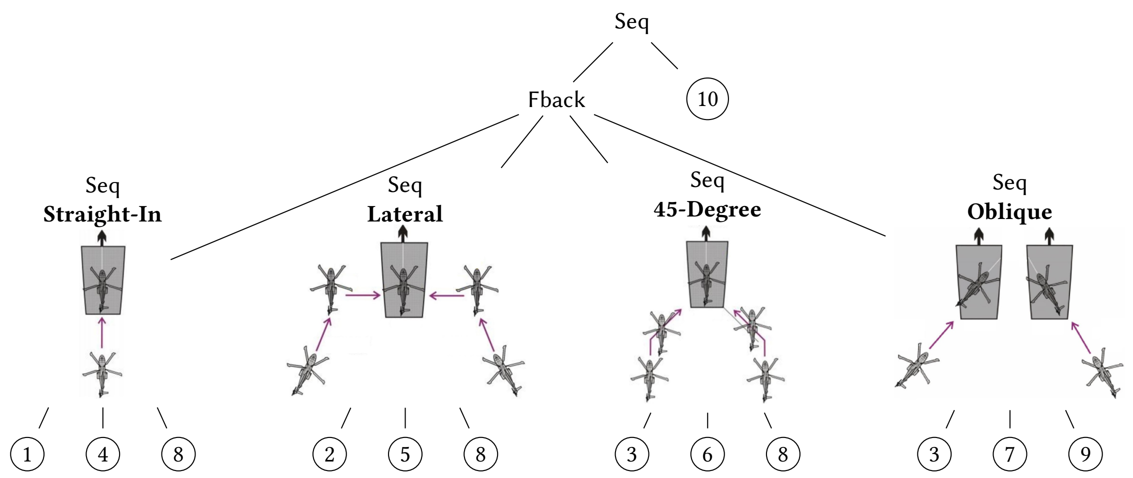


Figure 9: Segmentation of an oblique maneuver where the UAS deviates from the expected behavior. Ideally, the "Stay in position" segment should align with the "Maneuver: at position" line.

We now examine the potential failing segmentations for oblique maneuvers. Note that by Theorem 6, given a failing segmentation, we cannot draw conclusions about the segmentation of the original trace. We also see that the segmentation for such a maneuver is always a 45-Degree maneuver, depicted in Figure 2. In Figure 9, we show an alternative segmentation that is worse but uses an oblique landing approach. We received this alternative by using τ_t that was set greater than $|\sigma|$. Choosing such a τ_t ensures that the alternative segmentation uses different nodes for its segments. Fig. 9 shows that the *stay in position* is poorly assigned. We further examined the robustness values and concluded that the oblique heading relative to the ship expected by the TBT was not sufficiently implemented by the controller. Hence, it behaves "closer" to a 45-degree maneuver than an oblique maneuver. The results of both use-cases show that segmentation provides key insights into complex behaviors. Further, our experiments show that our current segmentation techniques can be used for large traces through a combination of subsampling the trace and approximation through lazy evaluation.

5 CONCLUSION

We have introduced temporal behavior trees (TBT) as a specification formalism, defined robustness semantics, and used it to segment a trace into subtraces that are associated with portions of the TBT specification using a dynamic programming algorithm that has



Leaf	Description	Temporal Formula
1	Behind ship	$\lozenge\ behind(p_s, p_{uas})$
2	Next to ship	$\lozenge\ nextTo(p_s, p_{uas})$
3	Diagonal behind ship	$\lozenge\ diagonalBehind(p_s, p_{uas})$
4	Stay in behind	$\square_{[0,1000]}\ (behind(p_s, p_{uas})\ \wedge\ heading(aligned, h_s, h_{uas})\ \wedge\ velocity(aligned, v_s, v_{uas}))$
5	Stay next to	$\square_{[0,1000]}\ (nextTo(p_s, p_{uas})\ \wedge\ heading(aligned, h_s, h_{uas})\ \wedge\ velocity(aligned, v_s, v_{uas}))$
6	Stay diagonal behind	$\square_{[0,1000]}\ (diagonalBehind(p_s, p_{uas})\ \wedge\ heading(aligned, h_s, h_{uas})\ \wedge\ velocity(aligned, v_s, v_{uas}))$
7	Stay diagonal behind	$\square_{[0,1000]}\ (diagonalBehind(p_s, p_{uas})\ \wedge\ heading(oblique, h_s, h_{uas})\ \wedge\ velocity(aligned, v_s, v_{uas}))$
8	Move to touchdown	$\lozenge\ (move\_to\_touchdown(above, p_s, p_{uas})\ \wedge\ heading(aligned, h_s, h_{uas}))$
9	Move to touchdown	$\lozenge\ (move\_to\_touchdown(above, p_s, p_{uas})\ \wedge\ heading(oblique, h_s, h_{uas}))$
10	Descending	$\lozenge\ (descended(p_{touchdown}, p_{uas}))$

Figure 10: Temporal behavior tree specifying landing maneuvers on a ship deck: straight-in, lateral, 45-degree, and oblique. They differ in their starting position and their heading relative to the ship. Computations of atomic propositions are omitted.

Logfile	δ	Time (ρ_δ, $\rho_\delta \circ \rho_{lazy}$) [s]	Chosen Maneuver (ρ_δ, $\rho_\delta \circ \rho_{lazy}$)	Worst Segment (ρ_δ, $\rho_\delta \circ \rho_{lazy}$)
45-Degree-WF	50	(4, < 0)	(45-Degree, 45-Degree)	(Descend 0.49, Descend 0.05)
45-Degree-WS	200	(< 0, < 0)	(45-Degree, 45-Degree)	(Descend 0.45, Descend 0.20)
Lateral-WF	100	(< 0, < 0)	(Lateral, Lateral)	(Descend 0.39, Descend 0.06)
Lateral-WS	100	(< 0, < 0)	(Lateral, Lateral)	(Descend 0.30, Descend 0.01)
Oblique-WF	25	(29, 29)	(45-Degree, 45-Degree)	(Move-to-pos -0.58, $-\infty$)
Oblique-WS	25	(28, 28)	(45-Degree, 45-Degree)	(Move-to-pos -0.34, $-\infty$)
Straight-in-WF	200	(< 0, < 0)	(Straight-in, Straight-in)	(Descend -0.60, $-\infty$)
Straight-in-WS	200	(< 0, < 0)	(Straight-in, Straight-in)	(Descend 0.22, Descend 0.22)

Table 1: Results of segmentation using the TBT in Fig. 10. The segmenation uses subsampling (Def. 8) alone and in combination with lazy evaluation (Def. 10). All logfiles represent realistic missions that took approximately two minutes (> $20,000$ entries). The results show that segmentations can be computed within seconds for practical use-cases.

been approximated through subsampling and lazy evaluation. We show on two use-cases that segmentation is a powerful tool when analyzing or debugging complex behaviors. We analyzed the performance of 25 human-operators flying a UAV waypoint mission. The second use-case reports on an autonomous landing of a UAV on a ship deck. We show that our algorithms can efficiently compute segmentations that help to better understand what parts of the mission succeeded or failed.

ACKNOWLEDGMENTS

This work was supported by the German federal aviation research program (LuFo ID: 20D2111C and ID: 20Q1963B), and the US National Science Foundation (NSF) under award number 1836900. S. Schirmer carried out this work as a member of the Saarbrücken Graduate School of Computer Science. We thank Alexander Donkels (DLR) for providing the data for the autonomous ship deck landing.

REFERENCES

[1] Rahib H. Abiyev, Nurullah Akkaya, and Ersin Aytac. 2013. Control of soccer robots using behaviour trees. In *9th Asian Control Conference, ASCC 2013, Istanbul, Turkey, June 23-26, 2013*. IEEE, 1–6. https://doi.org/10.1109/ASCC.2013.6606326

[2] Ezio Bartocci, Jyotirmoy Deshmukh, Alexandre Donzé, Georgios Fainekos, Oded Maler, Dejan Ničković, and Sriram Sankaranarayanan. 2018. Specification-based monitoring of cyber-physical systems: a survey on theory, tools and applications. *Lectures on Runtime Verification: Introductory and Advanced Topics* (2018), 135–175.

[3] Andreas Bauer, Martin Leucker, and Christian Schallhart. 2007. The good, the bad, and the ugly, but how ugly is ugly?. In *International Workshop on Runtime Verification*. Springer, 126–138.

[4] Andreas Bauer, Martin Leucker, and Christian Schallhart. 2011. Runtime verification for LTL and TLTL. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 20, 4 (2011), 1–64.

[5] Johan Van Benthem. 2010. *Modal Logic for Open Minds (Lecture Notes)*. CSLI Publications (Stanford University).

[6] Sooyung Byeon, Joonwon Choi, Yutong Zhang, and Inseok Hwang. 2023. Stochastic-Skill-Level-Based Shared Control for Human Training in Urban Air Mobility Scenario. *J. Hum.-Robot Interact.* (jun 2023). https://doi.org/10.1145/3603194 Just Accepted.

[7] Antonio Cau, Ben Moszkowski, and Hussein Zedan. 2006. Interval temporal logic. *URL: http://www. cms. dmu. ac. uk/˜ cau/itlhomepage/itlhomepage. html* (2006).

[8] Yuxiao Chen, James Anderson, Karanjit Kalsi, Aaron D. Ames, and Steven H. Low. 2021. Safety-Critical Control Synthesis for Network Systems With Control Barrier Functions and Assume-Guarantee Contracts. *IEEE Transactions on Control of Network Systems* 8, 1 (2021), 487–499. https://doi.org/10.1109/TCNS.2020.3029183

[9] Michele Colledanchise and Petter Ögren. 2018. *Behavior trees in robotics and AI: An introduction*. CRC Press.

[10] Giuseppe De Giacomo and Moshe Y. Vardi. 2013. Linear Temporal Logic and Linear Dynamic Logic on Finite Traces. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence* (Beijing, China) *(IJCAI '13)*. AAAI Press, 854–860.

[11] Jyotirmoy V Deshmukh, Alexandre Donzé, Shromona Ghosh, Xiaoqing Jin, Garvit Juniwal, and Sanjit A Seshia. 2017. Robust online monitoring of signal temporal logic. *Formal Methods in System Design* 51 (2017), 5–30.

[12] 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.

[13] Georgios E Fainekos and George J Pappas. 2006. Robustness of temporal logic specifications. In *International Workshop on Formal Approaches to Software Testing*. Springer, 178–192.

[14] Bernd Finkbeiner and Henny Sipma. 2004. Checking finite traces using alternating automata. *Formal Methods in System Design* 24 (2004), 101–127.

[15] Razan Ghzouli, Thorsten Berger, Einar Broch Johnsen, Swaib Dragule, and Andrzej Wąsowski. 2020. Behavior Trees in Action: A Study of Robotics Applications. In *Proceedings of the 13th ACM SIGPLAN International Conference on Software Language Engineering* (Virtual, USA) *(SLE 2020)*. Association for Computing Machinery, New York, NY, USA, 196–209. https://doi.org/10.1145/3426425.3426942

[16] Joseph Halpern, Zohar Manna, and Ben Moszkowski. 1983. A hardware semantics based on temporal intervals. In *Automata, Languages and Programming*, Josep Diaz (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 278–291.

[17] D. Harel and D. Peleg. 1985. Process logic with regular formulas. *Theoretical Computer Science* 38 (1985). https://doi.org/10.1016/0304-3975(85)90225-7

[18] Jie He, Ezio Bartocci, Dejan Ničković, Haris Isakovic, and Radu Grosu. 2022. DeepSTL: From English Requirements to Signal Temporal Logic. In *Proceedings of the 44th International Conference on Software Engineering* (Pittsburgh, Pennsylvania) *(ICSE '22)*. Association for Computing Machinery, New York, NY, USA, 610–622. https://doi.org/10.1145/3510003.3510171

[19] Keliang He, Morteza Lahijanian, Lydia E. Kavraki, and Moshe Y. Vardi. 2015. Towards manipulation planning with temporal logic specifications. In *IEEE International Conference on Robotics and Automation, ICRA 2015, Seattle, WA, USA, 26-30 May, 2015*. IEEE, 346–352. https://doi.org/10.1109/ICRA.2015.7139022

[20] Zhuochao He, Xuyang Zhang, Simon Jones, Sabine Hauert, Dandan Zhang, and Nathan F. Lepora. 2023. TacMMs: Tactile Mobile Manipulators for Warehouse Automation. *IEEE Robotics and Automation Letters* 8, 8 (2023), 4729–4736. https://doi.org/10.1109/LRA.2023.3287363

[21] Danying Hu, Yuanzheng Gong, Blake Hannaford, and Eric J Seibel. 2015. Semi-autonomous simulated brain tumor ablation with Raven II surgical robot using behavior trees. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 3868–3875.

[22] Hao Hu, Xiaoliang Jia, Kuo Liu, and Bingyang Sun. 2021. Self-Adaptive Traffic Control Model With Behavior Trees and Reinforcement Learning for AGV in Industry 4.0. *IEEE Transactions on Industrial Informatics* 17, 12 (2021), 7968–7979. https://doi.org/10.1109/TII.2021.3059676

[23] Matteo Iovino, Edvards Scukins, Jonathan Styrud, Petter Ögren, and Christian Smith. 2022. A survey of behavior trees in robotics and ai. *Robotics and Autonomous Systems* 154 (2022), 104096.

[24] Anja Johansson and Pierangelo Dell'Acqua. 2012. Emotional behavior trees. In *2012 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 355–362.

[25] Hadas Kress-Gazit, Georgios E. Fainekos, and George J. Pappas. 2009. Temporal-Logic-Based Reactive Mission and Motion Planning. *IEEE Transactions on Robotics* 25, 6 (2009), 1370–1381. https://doi.org/10.1109/TRO.2009.2030225

[26] Orna Kupferman, Giuseppe Perelli, and Moshe Y Vardi. 2016. Synthesis with rational environments. *Annals of Mathematics and Artificial Intelligence* 78, 1 (2016), 3–20.

[27] Martin Leucker and César Sánchez. 2007. Regular linear temporal logic. In *International colloquium on theoretical aspects of computing*. Springer, 291–305.

[28] Christopher D Marlin. 1979. *Coroutines: A Programming Methodology, a Language Design, and an Implementation*. Ph. D. Dissertation. University of Adelaide, Department of Computing Science.

[29] M. Mateas and A. Stern. 2002. A behavior language for story-based believable agents. *IEEE Intelligent Systems* 17, 4 (2002), 39–47. https://doi.org/10.1109/MIS.2002.1024751

[30] B. C. Moszkowski. 1998. Compositional Reasoning using Interval Temporal Logic and Tempura. In *Compositionality: The Significant Difference*, Willem-Paul de Roever, Hans Langmaack, and Amir Pnueli (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 439–464.

[31] Ana Lúcia De Moura and Roberto Ierusalimschy. 2009. Revisiting Coroutines. *ACM Trans. Program. Lang. Syst.* 31, 2, Article 6 (feb 2009), 31 pages. https://doi.org/10.1145/1462166.1462167

[32] Vasumathi Raman, Alexandre Donzé, Dorsa Sadigh, Richard M. Murray, and Sanjit A. Seshia. 2015. Reactive Synthesis from Signal Temporal Logic Specifications. In *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control* (Seattle, Washington) *(HSCC '15)*. Association for Computing Machinery, New York, NY, USA, 239–248. https://doi.org/10.1145/2728606.2728628

[33] Aurélien Rizk, Grégory Batt, François Fages, and Sylvain Soliman. 2008. On a Continuous Degree of Satisfaction of Temporal Logic Formulae with Applications to Systems Biology. In *Computational Methods in Systems Biology*. Springer Berlin Heidelberg, 251–268.

[34] Roni Rosner and Amir Pnueli. 1986. A Choppy Logic. In *Proceedings of the First Annual IEEE Symposium on Logic in Computer Science (LICS 1986)* (Cambridge, MA, USA). IEEE Computer Society Press, 306–313.

[35] Kirk Y. W. Scheper, Sjoerd Tijmons, Cornelis C. de Visser, and Guido C. H. E. de Croon. 2016. Behavior Trees for Evolutionary Robotics†. *Artificial Life* 22, 1 (02 2016), 23–48. https://doi.org/10.1162/ARTL_a_00192 arXiv:https://direct.mit.edu/artl/article-pdf/22/1/23/1665258/artl_a_00192.pdf

[36] T. Schmelz and R. Lantzsch. 2018. Abschlussbericht: F&T Studie - Pilotenassistenz für Schiffsdecklandungen (PiloDeck)[Final report: F&T Study - Pilot assitance for ship deck landing (PiloDeck)],. *Technical Note AHD-TN-ESPE-302-18* (2018).

[37] Bianca Isabella Schuchardt, Thomas Dautermann, Alexander Donkels, Stefan Krause, Niklas Peinecke, and Gunnar Schwoch. 2020. Maritime operation of an unmanned rotorcraft with tethered ship deck landing system. *CEAS Aeronautical Journal* 12, 1 (9 2020), 1–9. https://elib.dlr.de/140951/

[38] Aleksandr Sidorenko, Jesko Hermann, and Martin Ruskowski. 2022. Using Behavior Trees for Coordination of Skills in Modular Reconfigurable CPPMs. In *2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA)*. 1–8. https://doi.org/10.1109/ETFA52439.2022.9921558

[39] Dogan Ulus, Thomas Ferrère, Eugene Asarin, and Oded Maler. 2014. Timed Pattern Matching. In *Formal Modeling and Analysis of Timed Systems*, Axel Legay and Marius Bozga (Eds.). Springer International Publishing, Cham, 222–236.

[40] A. v. Perger, P. Gamper, and R. Witzmann. 2022. Behavior Trees for Smart Grid Control. *IFAC-PapersOnLine* 55, 9 (2022), 122–127. https://doi.org/10.1016/j.ifacol.2022.07.022 11th IFAC Symposium on Control of Power and Energy Systems CPES 2022.

[41] Pierre Wolper. 1983. Temporal logic can be more expressive. *Information and control* 56, 1-2 (1983), 72–99.

A PROOF OF THEOREM 1

We prove the Theorem 1 by structural induction on φ. We show that "For any trace σ and STL formula φ, we have $\sigma \models \varphi$ iff $\rho(\varphi, \sigma) \geq 0$".

Proof.

Base Cases:

Case $\varphi : p_i$ where $p_i \in AP$ and $|\sigma| > 0$,

$$\begin{aligned} & \sigma \models \varphi \\ \iff & f_i(\sigma(1)) \geq 0 \\ \overset{\text{Def. 2}}{\iff} & \rho(p_i, \sigma) \geq 0 \end{aligned}$$

Case $\varphi : p_i$ where $p_i \in AP$ and $|\sigma| = 0$,

$$\begin{aligned} & \sigma \not\models \varphi \\ \iff & f_i(\sigma(1)) < 0 \\ \overset{\text{Def. 2}}{\iff} & \rho(p_i, \sigma) < 0 \end{aligned}$$

Induction Step:

Case $\varphi : \neg\varphi_1$,

$$\begin{aligned} & \sigma \models \neg\varphi_1 \\ \iff & \sigma \not\models \varphi_1 \\ \overset{\text{IH}}{\iff} & \rho(\varphi_1, \sigma) < 0 \\ \overset{\text{Def. 2}}{\iff} & \rho(\neg\varphi_1, \sigma) > 0 \end{aligned}$$

Case $\varphi : \varphi_1 \wedge \varphi_2$,

$$\begin{aligned} & \sigma \models \varphi_1 \wedge \varphi_2 \\ \iff & \sigma \models \varphi_1 \wedge \sigma \models \varphi_2 \\ \overset{\text{IH}}{\iff} & \rho(\varphi_1, \sigma) \geq 0 \wedge \rho(\varphi_2, \sigma) \geq 0 \\ \iff & \min(\rho(\varphi_1, \sigma), \rho(\varphi_2, \sigma)) \geq 0 \\ \overset{\text{Def. 2}}{\iff} & \rho(\varphi_1 \wedge \varphi_2, \sigma) \geq 0 \end{aligned}$$

Case $\varphi : \varphi_1 \vee \varphi_2$,

$$\begin{aligned} & \sigma \models \varphi_1 \vee \varphi_2 \\ \iff & \sigma \models \varphi_1 \ \vee \ \sigma \models \varphi_2 \\ \overset{\text{IH}}{\iff} & \rho(\varphi_1, \sigma) \geq 0 \ \vee \ \rho(\varphi_2, \sigma) \geq 0 \\ \iff & \max(\rho(\varphi_1, \sigma), \rho(\varphi_2, \sigma)) \geq 0 \\ \overset{\text{Def. 2}}{\iff} & \rho(\varphi_1 \vee \varphi_2, \sigma) \geq 0 \end{aligned}$$

Case $\varphi : \lozenge_{[l,u]}(\varphi_1)$,

$$\begin{aligned} & \sigma \models \lozenge_{[l,u]}(\varphi_1) \\ \iff & \exists i \in [l,u],\ \sigma[i:] \models \varphi_1 \\ \overset{\text{IH}}{\iff} & \exists i \in [l,u],\ \rho(\varphi_1, \sigma[i:]) \geq 0 \\ \iff & \max_{i \in [l,u]} (\rho(\varphi_1, \sigma[i:])) \geq 0 \\ \overset{\text{Def. 2}}{\iff} & \rho(\lozenge_{[l,u]}(\varphi_1), \sigma) \geq 0 \end{aligned}$$

Case $\varphi : \square_{[l,u]}(\varphi_1)$,

$$\begin{aligned} & \sigma \models \square_{[l,u]}(\varphi_1) \\ \iff & \forall i \in [l,u],\ \sigma[i:] \models \varphi_1 \\ \overset{\text{IH}}{\iff} & \forall i \in [l,u],\ \rho(\varphi_1, \sigma[i:]) \geq 0 \\ \iff & \min_{i \in [l,u]} (\rho(\varphi_1, \sigma[i:])) \geq 0 \\ \overset{\text{Def. 2}}{\iff} & \rho(\square_{[l,u]}(\varphi_1), \sigma) \geq 0 \end{aligned}$$

Case $\varphi : \varphi_1 \ \mathcal{U}_{[l,u]} \ \varphi_2$,

$$\begin{aligned} & \sigma \models \varphi_1 \ \mathcal{U}_{[l,u]} \ \varphi_2 \\ \iff & \exists i \in [l,u], (\forall j \in [0, i-1], \sigma[j:] \models \varphi_1) \wedge \sigma[i:] \models \varphi_2 \\ \overset{\text{IH}}{\iff} & \exists i \in [l,u], (\forall j \in [0, i-1], \rho(\varphi_1, \sigma[j:]) \geq 0) \\ & \qquad \wedge \rho(\varphi_2, \sigma[i:]) \geq 0 \\ \iff & \max_{i \in [l,u]} \min(\rho(\varphi_2, \sigma[i:]), \min_{j \in [0,i-1]} \rho(\varphi_1, \sigma[j:])) \geq 0 \\ \overset{\text{Def. 2}}{\iff} & \rho(\varphi_1 \ \mathcal{U}_{[l,u]} \ \varphi_2, \sigma) \geq 0 \end{aligned}$$

□

B PROOF OF THEOREM 2

We prove the theorem by structural induction on $\mathcal{T}$. We show that "For any trace σ and TBT $\mathcal{T}$, $\sigma \models \mathcal{T}$ iff $\rho(\mathcal{T}, \sigma) \geq 0$".

Proof.

Base Case $\mathcal{T}$: $\mathsf{Leaf}(\varphi)$

$$\begin{aligned} & \sigma \models \mathsf{Leaf}(\varphi) \\ \iff & \sigma \models \varphi \\ \overset{\text{Thm. 1}}{\iff} & \rho(\varphi, \sigma) \geq 0 \end{aligned}$$

Induction Step:

Case $\mathcal{T}$: $\mathsf{Fback}([\mathcal{T}_1, \ldots, \mathcal{T}_k])$,

$$\begin{aligned} & \sigma \models \mathsf{Fback}([\mathcal{T}_1, \ldots, \mathcal{T}_k]) \\ \iff & \exists j \in \{1, \ldots, k\}, \exists i \in [0, |\sigma|-1], \sigma[i:] \models \mathcal{T}_j \\ \overset{\text{IH}}{\iff} & \exists j \in \{1, \ldots, k\}, \exists i \in [0, |\sigma|-1], \rho(\mathcal{T}_j, \sigma[i:]) \geq 0 \\ \iff & (\max_{j \in [1,k]} \max_{i \in [0,|\sigma|-1]} \rho(\mathcal{T}_j, \sigma[i:])) \geq 0 \\ \overset{\text{Def. 4}}{\iff} & \rho(\mathsf{Fback}([T_1, \ldots, T_k]), \sigma) \geq 0 \end{aligned}$$

Case $\mathcal{T}$: $\mathsf{Par}_M([\mathcal{T}_1, \ldots, \mathcal{T}_k])$,

$$\begin{aligned} & \sigma \models \mathsf{Par}_M([\mathcal{T}_1, \ldots, \mathcal{T}_k]) \\ \iff & \exists i_1, \ldots, i_M \in \{1, \ldots, k\}, \sigma \models \mathcal{T}_{i_1}, \ldots, \sigma \models \mathcal{T}_{i_M} \\ \overset{\text{IH}}{\iff} & \exists i_1, \ldots, i_M \in \{1, \ldots, k\}, \rho(\mathcal{T}_{i_1}, \sigma) \geq 0, \ldots, \rho(\mathcal{T}_{i_M}, \sigma) \geq 0 \\ \iff & max_M(\rho(\mathcal{T}_1, \sigma), \ldots, \rho(\mathcal{T}_k, \sigma)) \geq 0 \\ \overset{\text{Def. 4}}{\iff} & \rho(\mathsf{Par}_M([\mathcal{T}_1, \ldots, \mathcal{T}_k]), \sigma) \geq 0 \end{aligned}$$

Case $\mathcal{T}$: $\mathsf{Seq}([\mathcal{T}_1, \mathcal{T}_2])$,

$$\begin{aligned} & \sigma \models \mathsf{Seq}([\mathcal{T}_1, \mathcal{T}_2]) \\ \iff & \exists i \in [0, |\sigma|-1], \sigma[:i] \models \mathcal{T}_1 \wedge \sigma[i+1:] \models \mathcal{T}_2 \\ \overset{\text{IH}}{\iff} & \exists i \in [0, |\sigma|-1], \rho(\mathcal{T}_1, \sigma[:i]) \geq 0 \wedge \rho(\mathcal{T}_2, \sigma[i+1:]) \geq 0 \\ \iff & \max_{i \in [0,|\sigma|-1]} \min(\rho(\mathcal{T}_1, \sigma[:i]), \rho(\mathcal{T}_2, \sigma[i+1:])) \geq 0 \\ \overset{\text{Def. 4}}{\iff} & \rho(\mathsf{Seq}([\mathcal{T}_1, \mathcal{T}_2]), \sigma) \geq 0 \end{aligned}$$

Case $\mathcal{T}$: $\mathsf{Tout}_t(\mathcal{T}_1)$

$$\begin{aligned} & \sigma \models \mathsf{Tout}_t(\mathcal{T}_1) \\ \iff & \sigma[:\min(|\sigma|-1, t-1)] \models \mathcal{T}_1 \\ \overset{\text{IH}}{\iff} & \rho(\mathcal{T}_1, \sigma[:\min(|\sigma|-1, t-1)]) \geq 0 \\ \overset{\text{Def. 4}}{\iff} & \rho(\mathsf{Tout}_t(\mathcal{T}_1), \sigma) \geq 0 \end{aligned}$$

Case $\mathcal{T} : \bigstar_n(\mathcal{T}_1)$,

$$\begin{aligned} & \sigma \models \bigstar_n(\mathcal{T}_1) \\ \iff & \exists i \in [0, |\sigma|-1].\sigma[:i] \models \mathcal{T}_1 \wedge \sigma[i+1:] \models \bigstar_{n-1}(\mathcal{T}_1) \\ \overset{\text{IH}}{\iff} & \exists i \in [0, |\sigma|-1].\rho(\mathcal{T}_1, \sigma[:i]) \geq 0 \wedge \\ & \qquad \rho(\bigstar_{n-1}(\mathcal{T}_1), \sigma[i+1:]) \geq 0 \\ \iff & \max_{i \in [0,|\sigma|-1]} \min(\rho(\mathcal{T}_1, \sigma[:i]), \\ & \qquad \rho(\bigstar_{n-1}(\mathcal{T}_1), \sigma[i+1:])) \geq 0 \\ \overset{\text{Def. 4}}{\iff} & \rho(\mathrm{Seq}([\mathcal{T}_1, \bigstar_{n-1}(\mathcal{T}_1)]), \sigma) \geq 0 \\ \overset{\text{Def. 4}}{\iff} & \rho(\bigstar_n(\mathcal{T}_1), \sigma) \geq 0 \end{aligned}$$

□

C PROOF OF THEOREM 6

Let σ be a δ-stuttering trace for a given $\delta \geq 1$. Let $\mathcal{T}$ be a δ preserving TBT. Recall that (a) every occurrence of $\mathcal{U}_{[l,u]}$ has $l = 0$ and u divisible by δ; (b) every occurrence of $\Diamond_{[l,u]}$ and $\Box_{[l,u]}$ has l, u divisible by δ and (c) every occurrence of Tout_t has t divisible by δ.

Let $\mathcal{T}'$ be the TBT with each occurrence of $\bowtie_{[l,u]}$ replaced with $\bowtie_{[l/\delta, u/\delta]}$ for $\bowtie \in \{\mathcal{U}, \Diamond, \Box\}$ and every occurrence of Tout_t replaced by $\mathrm{Tout}_{t/\delta}$.

Let σ' be the subsampled trace where $\sigma'(i+1) = \sigma(i\delta+1)$ for $i \in \{0, \ldots, \frac{|\sigma|}{\delta} - 1\}$.

We say that a set of consecutive trace positions in σ forms a block B_i : $\{i\delta+1, \ldots, (i+1)\delta\}$. Thus, we can partition the indices of trace σ into $m = \frac{|\sigma|}{\delta}$ contiguous blocks $B_0, \ldots, B_{m-1}$.

First, we will prove for every TBT $\mathcal{T}$ that is δ-preserving and of the form $\mathrm{Leaf}(\varphi)$ that if φ holds at the first position of the block then it holds everywhere in the block (and vice-versa).

Lemma 3. *For a δ preserving TBT $\mathcal{T}$ of the form* $\mathrm{Leaf}(\varphi)$, *a δ-stuttering trace σ and index $i \geq 0$ such that $i\delta + 1 \leq |\sigma|$,*

$$\sigma[i\delta:] \models \varphi \textit{ iff } \sigma[i\delta+k:] \models \varphi,$$

for all $k \in \{0, \ldots, \delta-1\}$.

Note that the converse direction holds trivially since we can set $k = 0$.

Proof. Proof is by induction on the structure of the formula φ. Assume $0 \leq i < m$ and $0 \leq k \leq \delta - 1$.
Case φ is an atomic proposition p: True by definition of a δ-stuttering trace (Def. 8).
Case $\varphi : \varphi_1 \circ \varphi_2$ where $\circ \in \{\wedge, \vee\}$. Since we assume the result by induction on the subformulas φ_1, φ_2, the proof follows directly from that.
Case $\varphi : \neg\varphi_1$. By induction, we have

$$\sigma[i\delta:] \models \varphi_1 \text{ iff } \sigma[i\delta+k:] \models \varphi_1,$$

Therefore,

$$\sigma[i\delta:] \not\models \varphi_1 \text{ iff } \sigma[i\delta+k:] \not\models \varphi_1,$$

It follows that

$$\sigma[i\delta:] \models \neg\varphi_1 \text{ iff } \sigma[i\delta+k:] \models \neg\varphi_1,$$

Case $\varphi :\ \Diamond_{[l,u]}\varphi_1$.

$$\sigma[i\delta:] \models \Diamond_{[l,u]}\varphi_1 \text{ iff } \exists j \in [i\delta+l, i\delta+u]\ \sigma[j:] \models \varphi_1$$

Assume $\sigma[i\delta:] \models \Diamond_{[l,u]}\varphi_1$. Let $j+1 \in B_r$ for some block B_r : $\{r\delta+1, \ldots, (r+1)\delta\}$. The reason we consider $j+1$ is that $\sigma[j:]$ by convention begins at state $\sigma(j+1)$. Therefore, $j = r\delta + \hat{r}$ for $\hat{r} \in \{0, \ldots, \delta-1\}$

By induction hypothesis: $\sigma[r\delta+k:] \models \varphi_1$ for $k \in \{0, \ldots, \delta-1\}$. and $j - i\delta \in [l, u]$. Therefore, $\hat{r} + r\delta - i\delta \in [l, u]$. However, since l and u are divisible by δ and $\hat{r} < \delta$, we obtain

$$\frac{\hat{r}}{\delta} + r - i \in [\frac{l}{\delta}, \frac{u}{\delta}] \text{ or, equivalently, } r - i \in [\frac{l}{\delta}, \frac{u}{\delta}].$$

Therefore, $r\delta \in i\delta + [l, u]$. $\sigma[i\delta+k:] \models \Diamond_{[l,u]}\varphi_1$ for all k since $\sigma[r\delta+k:] \models \varphi_1$ and $(r\delta+k) - (i\delta+k) \in [l, u]$.

Conversely, if $\sigma[i\delta+k:] \models \Diamond_{[l,u]}\varphi_1$ for all k, then so does $\sigma[i\delta:]$.
Case $\varphi :\ \Box_{[l,u]}\varphi_1$. Proof is similar to the $\Diamond_{[l,u]}$ case.

Let $\sigma[i\delta:] \models \Box_{[l,u]}\varphi$. Therefore, for all $j \in i\delta + [l, u]$ we have

$$\sigma[j:] \models \varphi_1.$$

Consider block B_i : $\{i\delta+1, \ldots, (i+1)\delta\}$. Let $l = l'\delta$ and $u = u'\delta$. Then, we note by induction that all indices in the blocks:

$$B_{i+l'}, \ldots, B_{i+u'}$$

satisfy φ_1.

Therefore, for all $k \in \{0, \ldots, \delta-1\}$ for all $\hat{j} \in \{(i+l')\delta+1+k, \ldots, (i+u')\delta+1+k\}$, we have

$$\sigma[\hat{j}:] \models \varphi_1.$$

Therefore, $\sigma[i\delta+k:] \models \Box_{[l,u]}\varphi_1$.

The converse holds trivially.
Case $\varphi :\ \varphi_1\mathcal{U}_{[0,u]}\varphi_2$.

If $u = 0$ then φ is logically equivalent to φ_2 and the result immediately holds. Assume $u > 0$.

Let $\sigma[i\delta:] \models \varphi$. There exists $j \in [i\delta, i\delta+u]$ such that

$$\sigma[j:] \models \varphi_2$$

and for all $j' \in [i\delta, j-1]$

$$\sigma[j':] \models \varphi_1$$

Once again let B_r : $\{r\delta+1, \ldots, (r+1)\delta\}$ be the block containing $j+1$. Once again by induction, we have all indices in B_r satisfy φ_2. Also, all indices in the range $j' \in [i\delta, r\delta)$ satisfy $\sigma[j':] \models \varphi_1$.

If B_r is the same block as B_i then we note that

$$\sigma[i\delta+k:] \models \varphi_2 \text{ and thus } \sigma[i\delta+k:] \models \varphi_1\mathcal{U}_{[0,u]}\varphi_2.$$

Otherwise, we have $r > i$. Since $j - i\delta \leq u$ we have $j = r\delta + \hat{r}$ and $r\delta - i\delta + \hat{r} \leq u'\delta$. Dividing by δ, we get $r - i + \frac{\hat{r}}{\delta} \leq u'$ or $r - i \leq u'$. Therefore, $(r\delta - i\delta) \leq u$. Therefore, for all $k \in \{0, \ldots, \delta-1\}$, we have

$$r\delta - (i\delta + k) \leq u - k \leq u.$$

Therefore, we conclude that

$$\sigma[i\delta+k:] \models \varphi_1\mathcal{U}_{[l,u]}\varphi_2$$

since $\sigma[r\delta] \models \varphi_2$, $\sigma[j':] \models \varphi_1$ for all $j' \in [i\delta, r\delta)$ and $r\delta \in i\delta + k + [0, u]$.

□

To establish the main Theorem 6, we will need to establish the following lemma. Let φ be an STL formula that is δ preserving and φ' be obtained by transforming every interval $[l, u]$ associated with $\Diamond, \Box, \mathcal{U}$ into $[\frac{l}{\delta}, \frac{u}{\delta}]$.

Lemma 4. *For all i such that $i\delta < |\sigma|$, if $\sigma'[i:] \models \varphi'$ then $\sigma[i\delta:] \models \varphi$.*

Proof. Proof is once again by induction on the structure of the formula φ.
Case φ is an atomic proposition p: we have $\varphi' = \varphi$. The rest holds by definition of subsampling.
Case $\varphi \in \{\neg\varphi_1, \varphi_1 \vee \varphi_2, \varphi_1 \wedge \varphi_2\}$**:** holds by induction on the sub-formulas.
Case $\varphi : \Diamond_{[l\delta,u\delta]}\varphi_1$**:** We have $\varphi' : \Diamond_{[l,u]}\varphi_1'$. Therefore, $\sigma'[i:] \models \varphi'$ iff $\exists j \in [i+l, i+u]$ $\sigma'[j:] \models \varphi_1'$. By Ind. Hyp., $\sigma[j\delta:] \models \varphi_1$. Also, $j\delta \in i\delta + [l\delta, u\delta]$. Therefore, $\sigma[i\delta:] \models \Diamond_{[i\delta,j\delta]}\varphi_1$.
Case $\varphi : \Box_{[l\delta,u\delta]}\varphi_1$**:** We have $\varphi' : \Box_{[l,u]}\varphi_1'$. Therefore, $\sigma'[i:] \models \varphi'$ iff $\forall j \in [i+l, i+u]$ $\sigma'[j:] \models \varphi_1'$. By Ind. Hyp., $\sigma[j\delta:] \models \varphi_1$ for all $j \in i + [l,u]$. Using Lemma 3, we conclude that $\sigma[j\delta+k:] \models \varphi_1$ for all $j \in i + [l,u]$ and for all $k \in [0, \delta-1]$. Therefore, we conclude that $\sigma[i\delta:] \models \Box_{[l\delta,u\delta]}\varphi_1$.
Case $\varphi : \varphi_1 \mathcal{U}_{[0,u\delta]}\varphi_2$**:** We have $\varphi' : \varphi_1'\mathcal{U}_{[0,u]}\varphi_2'$. Therefore, $\sigma'[i:] \models \varphi'$ iff $\exists j \in [i, i+u]$ $\sigma'[j:] \models \varphi_2'$ and $\sigma'[j':] \models \varphi_1'$ for $j' \in [i,j)$. By Ind. Hyp., $\sigma[j\delta:] \models \varphi_2$ and $\sigma[j'\delta:] \models \varphi_1$. Lemma 3, we conclude that $\sigma[j'\delta + k:] \models \varphi_1$ for all $j' \in i + [0,j)$ and for all $k \in [0, \delta-1]$. Therefore, we conclude that $\sigma[i\delta:] \models \varphi_1 \mathcal{U}_{[l\delta,u\delta]}\varphi_2$. □

Proof of Theorem 6: Now we will prove a stronger version from which the required result follows directly.

Lemma 5. *If $\sigma'[i:j] \models \mathcal{T}'$ then $\sigma[i\delta : j\delta+\delta-1] \models \mathcal{T}$*

Proof. Let $\sigma'[i:j] \models \mathcal{T}'$. If $\sigma'[i:j]$ is empty then so is $\sigma[i\delta : j\delta+\delta-1]$ and the statement holds trivially.
Proof is on the structure of the $\mathcal{T}$.
Case $\mathcal{T} = \mathsf{Leaf}(\varphi)$**:** Follows directly from Lemma 4 applied to the sub-trace $\sigma[i\delta : j\delta+\delta-1]$ which is also a delta-stuttering trace.
Case $\mathcal{T} : \mathsf{Seq}([\mathcal{T}_1, \mathcal{T}_2])$.

$$\begin{aligned}&\sigma'[i:] \models \mathsf{Seq}([\mathcal{T}_1', \mathcal{T}_2']) \\ &\qquad\qquad \Rightarrow \quad \exists\, u \geq i,\ \sigma[i:u] \models \mathcal{T}_1',\ \sigma[u+1:] \models \mathcal{T}_2'\end{aligned}$$

Note that $\sigma[i\delta : u\delta+\delta-1]$ and $\sigma[(u+1)\delta:]$ are δ stuttering traces and by induction hypothesis, we have

$$\sigma[i\delta : u\delta+\delta-1] \models \mathcal{T}_1 \text{ and } \sigma[(u+1)\delta:] \models \mathcal{T}_2\,.$$

Therefore, $\sigma \models \mathsf{Seq}(\mathcal{T}_1, \mathcal{T}_2)$.
Case $\mathcal{T} : \mathsf{Fback}([\mathcal{T}_1, \ldots, \mathcal{T}_k])$.
$\sigma'[i:j] \models \mathsf{Fback}([\mathcal{T}_1', \ldots, \mathcal{T}_k'])$. It follows that there exists $r \geq i$ and $l \in [1,k]$ such that

$$\sigma'[r:j] \models \mathcal{T}_l'\,.$$

By induction, $\sigma[r\delta : j\delta+\delta-1] \models \mathcal{T}_l$. Therefore, $\sigma[i\delta : j\delta+\delta-1] \models \mathsf{Fback}([\mathcal{T}_1, \ldots, \mathcal{T}_k])$.
Case $\mathcal{T} : \mathsf{Par}_M([\mathcal{T}_1, \ldots, \mathcal{T}_k])$.
We have $\sigma'[i:j] \models \mathsf{Par}_M([\mathcal{T}_1', \ldots, \mathcal{T}_k'])$ iff there are M subtrees $\mathcal{T}_{i_1}', \ldots, \mathcal{T}_{i_M}'$ such that

$$\sigma'[i:j] \models \mathcal{T}_{i_j}'\,.$$

By induction $\sigma[i\delta : j\delta+\delta-1] \models \mathcal{T}_{i_j}$ and thus

$$\sigma[i\delta : j\delta+\delta-1] \models \mathsf{Par}_M([\mathcal{T}_1, \ldots, \mathcal{T}_k])\,.$$

Case $\mathcal{T} : \mathsf{Tout}_t(\mathcal{T}_1)$.
We have $\sigma'[i:j] \models \mathsf{Tout}_t\mathcal{T}_1'$. Assume that $j - i + 1 \geq t$.
$\sigma'[i:i+t-1] \models \mathcal{T}_1'$. By induction, $\sigma[i\delta : i\delta+(t-1)\delta+\delta-1] \models \mathcal{T}_1$. Therefore, $\sigma[i\delta : t\delta-1] \models \mathsf{Tout}_{t\delta}(\mathcal{T}_1)$.

Case $\mathcal{T} : \bigstar_n(\mathcal{T}_1)$.
Let $\sigma'[i:j] \models \bigstar_n(\mathcal{T}_1)$. There exists indices $-1 = i_1, \ldots, i_k = j$ for $k \leq n+1$ such that $\sigma'[i_1+1 : i_2] \models \mathcal{T}_1'$, $\sigma[i_2+1 : i_3] \models \mathcal{T}_1'$, $\cdots$, $\sigma[i_{k-1}+1 : i_k] \models \mathcal{T}_1'$. Therefore, for each l, $\sigma[(i-l+1)\delta : i_{l+1}\delta+\delta-1] \models \mathcal{T}_1$. Therefore, $\sigma[i\delta : j\delta+\delta-1] \models \bigstar_n(\mathcal{T}_1)$. □

Theorem 6 follows directly.