Continuous-Time Behavior Trees as Discontinuous Dynamical Systems

Christopher I. Sprague, *Student Member, IEEE*, and Petter Ögren, *Senior Member, IEEE*

***Abstract*—Behavior trees represent a hierarchical and modular way of combining several low-level control policies into a high-level task-switching policy. Hybrid dynamical systems can also be seen in terms of task switching between different policies, and therefore several comparisons between behavior trees and hybrid dynamical systems have been made, but only informally, and only in discrete time. A formal continuous-time formulation of behavior trees has been lacking. Additionally, convergence analyses of specific classes of behavior tree designs have been made, but not for general designs. In this letter, we provide the first continuous-time formulation of behavior trees, show that they can be seen as discontinuous dynamical systems (a subclass of hybrid dynamical systems), which enables the application of existence and uniqueness results to behavior trees, and finally, provide sufficient conditions under which such systems will converge to a desired region of the state space for general designs. With these results, a large body of results on continuous-time dynamical systems can be brought to use when designing behavior tree controllers.**

***Index Terms*—Autonomous systems, behavior trees, stability of hybrid systems, switched systems.**

I. Introduction

BEHAVIOR trees (BTs) are a way to combine a set of controllers (policies) into higher-level controllers in a hierarchical and modular way. In this letter, we give the first continuous-time representation of BTs and provide sufficient conditions for convergence of general BTs.

Modularity is a key tool to handle complexity in software systems, as it enables different components to be developed and tested individually, and BTs have been shown to be optimally modular in comparison to other decision structures [1]. Hierarchical modularity, where each module may contain submodules, is also beneficial since a single level of modules in a large system either leads to very large and complex modules, or a very large number of smaller modules. Additionally, a hierarchical structure is more natural in many applications, as many tasks can be divided into subtasks in a hierarchical way, such as when a robot has to fetch an object, which might include subtasks such as navigation, door opening, object grasping, and so on.

Improved modularity is the reason that BTs were conceived in the first place [2] as an equally expressive [3] alternative to finite-state machines (FSMs) in the design of non-player characters in video games. In this virtual setting, the world is predictable by design and many low-level policies can be developed with relative ease. Thus, game developers started to put together large sets of low-level policies earlier than robot developers and therefore had a stronger need for modular tools. However, the interest in BTs from the robotics community has increased over time and they are now used in both open-source middleware, such as the Robotic Operating System (ROS)[1] and innovative industry software from Boston Dynamics[2] and Nvidia.[3]

Even though there is an increasing interest in BTs from the robotics and AI communities (see the recent survey in [4] with over 180 papers) there is still no continuous-time formulation available. The need for such a formulation is clear from the fact that almost all major branches of control theory, from linear systems to optimal control, have been developed for both continuous-time and discrete-time systems, but BTs have so far only had a discrete-time formulation. With the proposed continuous-time model, continuous-time control theory results, such as sliding mode control, can now be used to analyze BT designs. To date, the only efforts towards continuous-time models have either been informal comparisons of BTs and hybrid dynamical systems (HDS), considering discrete-time BTs and discrete-time HDS, or different ways of doing event-based ticking, or letting the tick frequency go to infinity [5]–[8].

A key topic in control theory is stability and convergence to a particular equilibrium point, or region of the state space. For a BT, this translates to reaching the so-called success region, a state where the BT returns success. Important results on sufficient conditions for convergence to the success region

Manuscript received September 6, 2021; revised November 11, 2021; accepted December 2, 2021. Date of publication December 10, 2021; date of current version December 23, 2021. This work was supported by SSF through the Swedish Maritime Robotics Centre (SMaRC) under Grant IRC15-0046. Recommended by Senior Editor L. Menini. *(Corresponding author: Christopher I. Sprague.)*

The authors are with the Robotics, Perception and Learning Laboratory, School of Electrical Engineering and Computer Science, Royal Institute of Technology (KTH), 100 44 Stockholm, Sweden (e-mail: sprague@kth.se).

Digital Object Identifier 10.1109/LCSYS.2021.3134453

[1] https://navigation.ros.org/configuration/packages/configuring-bt-navigator.html

[2] https://dev.bostondynamics.com/docs/concepts/autonomy/missions_service

[3] https://docs.nvidia.com/isaac/isaac/packages/behavior_tree/doc/behavior_trees.html

have been presented in [9], [10], but in both cases the analysis was limited to a particular subclass of BTs. In this letter we propose sufficient conditions that can be use to analyze any BT design.

The main contributions of this letter are as follows. We provide the first formal formulation of BTs in continuous time (Definition 1). We show that the proposed formulation can be seen as a discontinuous dynamical system (DDS) (Theorem 2), with corresponding results regarding existence and uniqueness (Theorem 3). We provide sufficient conditions under which a BT execution will converge to a desired region of the state space (Theorem 4).

The organization of this letter is as follows. In Section II, we discuss how our contributions differ from those presented in related work. In Section III, we provide a brief overview of tools for analyzing ordered trees and results regarding DDSs. Then, in Section IV, we formulate continuous-time BTs and connect them to DDSs in Section V. Finally, in Section VI, we present a convergence proof and in Section VII, we state our conclusions.

II. Related Work

In this section, we will describe related work from a number of different aspects.

Continuous-Time: In [6], a continuous-time BT is informally described as a discrete-time BT with an infinite tick rate, as a means to compare BTs to HDSs. In [8], instead of querying behaviors at a certain tick rate, behaviors run continuously and notify superior behaviors when their status changes. *Our work addresses the same problems; however, our work does so on the basis of a formal state space definition of continuous-time BTs (Definition 1).*

Hybrid Dynamical Systems: The first comparison of BTs to HDSs appears to have been made in [5]. Therein, it was described how BTs modularly represent HDSs and implicitly encode explicit state transitions through its tree structure. This discussion continued along the same lines in [7] and equivalence notions between discrete-time BTs and HDSs were presented in [6].

In these works, the interpretation of an HDS is such that a discrete state determines which behavior to use. However, as we will show, a BT is aptly described by a DDS [11], where the state's presence in certain regions solely determines which behavior is used. *Thus, we go beyond related work by not only showing that BTs more closely correspond to DDSs [11], but we also do this formally (Theorem 2). As a result, we also address existence and uniqueness of solutions (Theorem 3).*

Convergence Analysis: It was shown in [7] that the composition of behaviors in Fallback BTs is similar to the idea of sequential composition [12]. Therein, sufficient conditions for convergence to a goal state were presented formally in terms of the attraction region of individual behaviors. These concepts were applied in [13] to guarantee BT performance in the presence of black-box controllers.

A version of BTs called Robust Logical-Dynamical Systems was proposed in [9], which uses an Implicit-Sequence BT structure like in [7]. Therein, they show convergence in the presence of uncontrolled behavior changes. *Our work is related to all of the above in that we prove convergence in BTs (Theorem 4); however, our work is different in the sense that the results can be applied to general BT structures, not just special classes.*

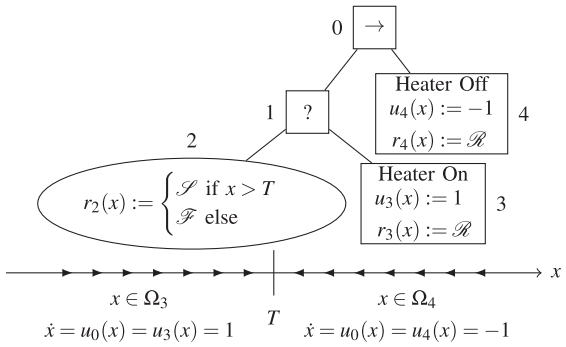

Fig. 1. A thermostat state-feedback controller modeled by a BT (top), and the phase portrait of its corresponding discontinuous dynamical system $\dot{x} = f(x, u_0(x)) = u_0(x)$ (bottom). If $x > T$ then $x \in \Omega_4$ and $\dot{x} = u_4(x) = -1$. Conversely, if $x \leq T$ then $x \in \Omega_3$ and $\dot{x} = u_3(x) = 1$, see Theorem 2.

A concept of [12] not used in the above works is the "prepares graph", a directed graph of transitions induced by the composition of policies. In [12], this graph is used to construct a totally ordered subgraph of policies that lead to the goal state. This construction was extended in [14] to allow for multiple controllers in the subgraph to overlap in order to attain more flexibility in the presence of disturbances, thereby forming a partially ordered subgraph. *We will use this notion of a prepares graph as a tool to prove the convergence of general BTs.*

III. Preliminaries

In this section we will first describe how two partial orders can be used for analyzing ordered trees, and then present some results on DDSs.

A. Ordered Trees

As we will see below, BTs are ordered trees, and as was discussed in [15], ordered trees can either be seen as graphs, as drawn in Fig. 1, or as a set of vertices with two partial orders, the so-called parent and sibling orders.

A directed graph is often defined in terms of $G = (V, E)$, where V is the vertices and $E \subset V^2$ is the edges. If the graph has no cycles and no two distinct paths from a starting vertex meet at the same ending vertex, it is called a tree; if one vertex is designated as the root, it is called rooted. Given a root, the usual concepts of parent/child can be applied to each edge, with the parent being closer to the root and the child further away. To create an ordering between siblings (children of the same parent) the vertices can be embedded in a plane (as drawn on a paper) and the order given by clockwise or left/right positions. In Fig. 1, the root would be vertex 0, and its two children vertex 1 and 4 (in that order) and so on.

In this letter, we will use the graph model for BTs, but we will also make use of order theory for analyzing ordered

trees, as described in [15]. As we will show, this formulation will support the analysis. We now use $(V, \leq_S, \leq_P)$ to define the tree, where V is the vertex set as above, and $\leq_S, \leq_P$ are two partial orders on V, called the sibling and parent orders, respectively.

A partial order $\leq$ on a set is a homogeneous binary relation $\leq \subset V^2$ (if $(x, y) \in \leq$ we write $x \leq y$) that is reflexive ($\forall x \in V : x \leq x$), antisymmetric ($\forall x, y \in V : (x \leq y) \wedge (y \leq x) \implies x = y$), and transitive ($\forall x, y, z \in V : (x \leq y) \wedge (y \leq z) \implies x \leq z$). The order is partial, since two elements x, y might not satisfy $x \leq y$ or $y \leq x$. If so, x, y are said to be incomparable by $\leq$. If all elements are comparable, the order is said to be a total order, instead of a partial order. We write $x < y$ if $x \leq y$ and $x \neq y$, and for the reversed order $\geq$ we write $y \geq x$ if $x \leq y$.

In Fig. 1, we have that $1 \leq_S 4$, since 1 and 4 are siblings and 1 is to the left of 4. Note that 0 and 1 are incomparable by $\leq_S$, since they have no sibling relation. Instead, they are comparable by $\leq_P$, with $0 \leq_P 1$. Furthermore, 0 and 3 are comparable by $\leq_P$, with $0 \leq_P 3$ by transitivity, but 0 and 3 are incomparable by $\leq_S$.

We can also combine orders into new orders as

$$\leq_A \circ \leq_B := \left\{(x, z) \in V^2 | \exists y \in V : (x \leq_A y) \wedge (y \leq_B z)\right\}. \quad (1)$$

In this way, we can define a generalized uncle relation from the sibling and parent relations as $<_{LU} := <_S \circ \leq_P$ (left uncle) $>_{RU} := >_S \circ \leq_P$ (right uncle). These relations include several steps in both sibling and parent directions, thus including siblings, uncles, great uncles, great-great uncles, and so on. In Fig. 1, we have that $4 >_{RU} 2$ and $4 >_{RU} 3$ because 4 is a right uncle of 2 and 3.

Independently of the graph or ordered set representations, we will use the parent map $p : V \to V$, mapping a vertex to its parent.

B. Dynamical Systems Theory

In this section, we will remind readers of a result from [11] on the existence and uniqueness of the solutions to DDSs. The notation used here will be used in the following sections to show how BTs fit into this formalism.

Theorem 1 (Existence and Uniqueness [11, Proposition 5, p.53]): Let $X : \mathbb{R}^n \to \mathbb{R}^n$ be a piecewise continuous vector field, with $\mathbb{R}^n = D_1 \cup D_2$. Let $S_X = \partial D_1 = \partial D_2$, where ∂ is the boundary operator, be the set of points at which X is discontinuous, and assume that S_X is a C^2-manifold. Furthermore, assume that, for $i \in \{1, 2\}$, $X_{|\bar{D}_i}$ is continuously differentiable on D_i and $X_{|\bar{D}_1} - X_{|\bar{D}_2}$ is continuously differentiable on S_X, where $X_{|\bar{D}_i}$ is the continuous extension of the restriction of X to $\bar{D}_i$. If, for each $x \in S_X$, either $X_{|\bar{D}_1}$ points into D_2 or $X_{|\bar{D}_2}$ points into D_1, there will exist a unique Filippov solution to $\dot{x} = X(x)$ starting from each initial condition.

IV. Continuous-Time BTs

In this section, we will define continuous-time BTs, and see how the example of Fig. 1 forms a continuous-time controller.

As noted above, BTs are a hierarchical and modular way of combining controllers into new controllers. In this letter we let all controllers be state-feedback controllers, i.e., functions from the state space $\mathbb{R}^n$ to some control space $\mathbb{R}^m$. If one wants to include some internal dynamics, such as a Kalman filter, in the controller, the state space can be extended.

Definition 1 (Behavior Tree): A function $\mathscr{T}_i : \mathbb{R}^n \to \mathbb{R}^m \times \{\mathscr{R}, \mathscr{S}, \mathscr{F}\}$, defined as

$$\mathscr{T}_i(x) := (u_i(x), r_i(x)), \quad (2)$$

where $i \in V$ is an index, $u_i : \mathbb{R}^n \to \mathbb{R}^m$ is a controller, and $r_i : \mathbb{R}^n \to \{\mathscr{R}, \mathscr{S}, \mathscr{F}\}$ is a metadata function, describing the progress of the controller in terms of the outputs: running ($\mathscr{R}$), success ($\mathscr{S}$), and failure ($\mathscr{F}$). Define the metadata regions for $x \in \mathbb{R}^n$ as the running, success, and failure regions:

$$R_i := \{x : r_i(x) = \mathscr{R}\},$$
$$S_i := \{x : r_i(x) = \mathscr{S}\}, \quad F_i := \{x : r_i(x) = \mathscr{F}\}, \quad (3)$$

respectively, which are pairwise disjoint and cover $\mathbb{R}^n$.

The metadata can intuitively be interpreted as follows. If $x \in S_i$, $\mathscr{T}_i$ has either succeeded with whatever it was supposed to do (such as opening a door), or the goal was already achieved to begin with (the door was open). Either way, it might make sense to execute another controller to achieve some other goal (perhaps a goal that was intended to be achieved after opening the door).

If $x \in F_i$, $\mathscr{T}_i$ has either failed (the door to be opened turned out to be locked), or has no chance of succeeding (the door is out of reach from the current position). Either way, it might make sense to execute another controller (either to open the door in some other way or to achieve a higher-level goal in a way that does not involve opening the door).

If $x \in R_i$, it is too early to determine if $\mathscr{T}_i$ will succeed or fail. In most cases, it makes sense to continue executing $\mathscr{T}_i$, but it could also be reasonable to change the controller if some other action is more important (e.g., low battery level indicates the need for recharging).

Definition 2 (Continuous BT Execution): Given some dynamical system $f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ that is to be controlled, and assuming the root of the BT is $\mathscr{T}_0$ (has index 0), we have

$$\dot{x} = f(x, u_0(x)), \quad (4)$$

where $u_0(x)$ is given by (2).

Below we will describe the properties of this execution, and in particular show that it can be seen as a DDS, with corresponding results regarding the existence and uniqueness of solutions.

As described above, knowing if a lower-level controller failed, succeeded, or is still trying (running) is crucial for a higher-level controller to decide if another sequence should be initiated, or if some kind of fallback action needs to be invoked to achieve the desired outcome. These two cases are captured by the two fundamental BT composition types: Sequence and Fallback. The result of these behavior compositions is simply another BT that satisfies (2). This is what gives BTs their hierarchical modularity.

A Sequence is used to combine subtrees that are to be executed in order, where each one requires the *success* of the previous action. If any subtree fails, the whole sequence fails.

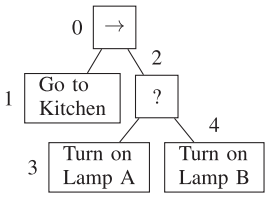

Fig. 2. An example BT containing two composition nodes: a Sequence (node 0) and a Fallback (node 2).

In Fig. 2, node 0 is a Sequence. First, node 1 is executed to get into the kitchen, and then node 2 is executed to turn one of the lamps on. But it only makes sense to try turning the lamps on if the action of moving to the kitchen succeeds. Formally, a Sequence is defined as follows.

Definition 3 (Sequence): A function *Seq* that composes an arbitrarily finite sequence of $M \in \mathbb{N}$ BTs into a new BT as

$$Seq[\mathscr{T}_1, \dots, \mathscr{T}_M](x) := \begin{cases} \mathscr{T}_M(x) & \text{if } x \in S_1 \cap \dots S_{M-1} \\ \vdots & \vdots \\ \mathscr{T}_2(x) & \text{else-if } x \in S_1 \\ \mathscr{T}_1(x) & \text{else.} \end{cases} \tag{5}$$

If $\mathscr{T}_i = Seq[\mathscr{T}_1, \dots, \mathscr{T}_M]$, then $j, k \in \{1, \dots, M\}$ are the children of i, such that $p(j) = i$, and are related as siblings, by $j \leq_S k$, if $j \leq k$.

As can be seen in (5), a subtree $\mathscr{T}_i$ is only executed if the state is in the success region of the siblings to the left $\mathscr{T}_j, j < i$.

A Fallback on the other hand only executes the next subtree if the previous one *fails*. If any subtree succeeds, the Fallback returns success, but it only returns failure if all subtrees fail. In Fig. 2, node 2 is a Fallback, and the two subtrees correspond to turning on either lamp A or lamp B.

Definition 4 (Fallback): A function *Fal* that composes an arbitrarily finite sequence of $M \in \mathbb{N}$ BTs into a new BT as

$$Fal[\mathscr{T}_1, \dots, \mathscr{T}_M](x) := \begin{cases} \mathscr{T}_M(x) & \text{if } x \in F_1 \cap \dots F_{M-1} \\ \vdots & \vdots \\ \mathscr{T}_2(x) & \text{else-if } x \in F_1 \\ \mathscr{T}_1(x) & \text{else.} \end{cases} \tag{6}$$

If $\mathscr{T}_i = Fal[\mathscr{T}_1, \dots, \mathscr{T}_M]$, then $j, k \in \{1, \dots, M\}$ are the children of i, such that $p(j) = i$, and are related as siblings, by $j \leq_S k$ if $j \leq k$.

The metadata regions (3) of the Sequence and Fallback compositions are given by the definition, but can also be explicitly computed in terms of the children regions and the orders $<_S, <_P$ as follows.

Lemma 1: The metadata regions of a Sequence $\mathscr{T}_i$ can be computed from the children metadata regions as follows:

$$R_i = \bigcup_{p(j)=i} \left(R_j \bigcap_{k<_S j} S_k \right),$$
$$S_i = \bigcap_{p(j)=i} S_j, \quad F_i = \bigcup_{p(j)=i} \left(F_j \bigcap_{k<_S j} S_k \right). \tag{7}$$

Proof: A straightforward application of (3) and (5). The running region of the sequence is the running region of the first child and the intersection of the success region of the first child with the running region of the second child and so on. The failure region works similarly, whereas the success region is the intersection of all the children success regions, as the sequence requires all children to succeed to return success. ■

Lemma 2: The metadata regions of a Fallback $\mathscr{T}_i$ can be computed from the children metadata regions as follows

$$R_i = \bigcup_{p(j)=i} \left(R_j \bigcap_{k<_S j} F_k \right),$$
$$S_i = \bigcup_{p(j)=i} \left(S_j \bigcap_{k<_S j} F_k \right), \quad F_i = \bigcap_{p(j)=i} F_j. \tag{8}$$

Proof: A straightforward application of (3) and (6). The running region is similar as for the Sequence above. The success region is similar to the running region, but the failure region is different since it requires all children to fail before returning failure. ■

V. BTs as Discontinuous Dynamical Systems

We need to show that the BT execution of (4) can be seen as a DDS. Thus we need to identify the operating regions Ω_i of the BT, i.e., the regions where the root BT executes a particular subtree $\mathscr{T}_0 = \mathscr{T}_i$. As we will see, the Ω_i will depend on both the subtree $\mathscr{T}_i$ itself, and its place in the surrounding BT. But, before we can define the operating region Ω_i we need to define the influence region I_i and the success and failure pathways $\mathfrak{S}, \mathfrak{F}$.

Informally, the influence region I_i is the region where the design of $\mathscr{T}_i$ influences the execution of $\mathscr{T}_0$, either by returning, e.g., failure so another node executes or by executing itself (thus we will have $I_i \supset \Omega_i$).

We will be using the so-called left uncle (LU) order $<_{LU} := <_S \circ \leq_P$ defined in Section III. Note that $\mathscr{T}_j : j <_{LU} i$ are left siblings of either i or any ancestors of i. For a state to be in I_i it needs to be in the success region of the left uncles that have a Sequence as a parent, and in the failure region of the left uncles that have a Fallback as a parent. Formally we write the following.

Definition 5 (Influence Region): A subset of the state space defined for $\mathscr{T}_i$ as

$$I_i := \bigcap_{\substack{j<_{LU} i \\ \mathscr{T}_{p(j)} \text{is Seq}}} S_j \quad \cap \quad \bigcap_{\substack{j<_{LU} i \\ \mathscr{T}_{p(j)} \text{is Fal.}}} F_j \tag{9}$$

In the example of Fig. 2, assuming the state space is $\mathbb{R}^n$, we have that $I_0 = \mathbb{R}^n$, $I_1 = \mathbb{R}^n$, $I_2 = S_1$, $I_3 = S_1$, and $I_4 = S_1 \cap F_3$. Thus, a change in $\mathscr{T}_1$ can influence $\mathscr{T}_0$ in any part of the state space, but a change in $\mathscr{T}_4$ can only influence $\mathscr{T}_0$ if $x \in S_1 \cap F_3$, i.e., if going to the kitchen was successful and turning on lamp A failed.

If the state is in I_i and $\mathscr{T}_i$ returns running, it will execute. But, it will also execute in the case when $\mathscr{T}_i$ returns success or failure and that same metadata is progressed all the way up to the root. Thus we need to identify what subtrees are on the so-called success and failure pathways. We now make use of the right uncle (RU) order that was also defined in Section III,

$>_{RU} := >_S \circ \leq_P$. Similarly, $\mathscr{T}_j : j >_{RU} i$ are right siblings of either i or any ancestors of i.

Informally, success pathways are vertices i such that there are no right uncles, with Sequence parents, that can take over the execution when $\mathscr{T}_i$ returns success. Similarly, failure pathways are vertices i such that there are no right uncles, with Fallback parents, that can take over the execution when $\mathscr{T}_i$ returns failure. We call them pathways since if i is on the pathway then so is every other vertex on the path from i to the root. Formally, we write the following.

Definition 6 (Success and Failure Pathways):

$$\mathfrak{S} := \left\{ i \in V |\ \nexists j \in V : (j >_{RU} i) \wedge \left(\mathscr{T}_{p(j)} \text{ is } Seq\right) \right\} \quad (10)$$

$$\mathfrak{F} := \left\{ i \in V |\ \nexists j \in V : (j >_{RU} i) \wedge \left(\mathscr{T}_{p(j)} \text{ is } Fal\right) \right\}, \quad (11)$$

respectively.

In the example of Fig. 2, we have that $\mathfrak{S} = \{0, 2, 3, 4\}$, since success from these nodes leads to success of the entire BT, and only success in going to the kitchen leads to other actions. Similarly, $\mathfrak{F} = \{0, 1, 2, 4\}$, since failure from these nodes leads to failure of the entire BT, and only a failure in turning on lamp A can be handled (by turning on lamp B).

We are now ready to define the operating regions.

Definition 7 (Operating Region): A subset of the state space defined for $\mathscr{T}_i$ as

$$\Omega_i := \begin{cases} I_i \cap (R_i \cup S_i \cup F_i) = I_i & \text{if } i \in \mathfrak{S} \cap \mathfrak{F} \\ I_i \cap (R_i \cup S_i) & \text{else-if } i \in \mathfrak{S} \\ I_i \cap (R_i \cup F_i) & \text{else-if } i \in \mathfrak{F} \\ I_i \cap R_i & \text{else.} \end{cases} \quad (12)$$

In the example of Fig. 2, we have that $\Omega_0 = \mathbb{R}^n$, $\Omega_1 = R_1 \cup F_1$, $\Omega_2 = S_1 \cap (R_2 \cup S_2)$, $\Omega_3 = S_1$, $\Omega_4 = S_1 \cap F_3$.

We will now show that a BT's operating region is partitioned by its childrens' operating regions.

Lemma 3: Operating regions of siblings are pairwise disjoint, $\Omega_i \cap \Omega_j = \emptyset$ for all $i <_S j$, and cover their parent's operating region, $\Omega_i = \bigcup_{p(j)=i} \Omega_j$.

Proof: As shown in [7], compositions can be expressed as follows: $Seq[\mathscr{T}_1, \mathscr{T}_2, \mathscr{T}_3] = Seq[\mathscr{T}_1, Seq[\mathscr{T}_2, \mathscr{T}_3]]$ and $Fal[\mathscr{T}_1, \mathscr{T}_2, \mathscr{T}_3] = Fal[\mathscr{T}_1, Fal[\mathscr{T}_2, \mathscr{T}_3]]$. Thus, it is sufficient to analyze the case of two children.

Let $0, 1, 2 \in V$ such that $1 <_S 2$ and $p(1) = p(2) = 0$. We will now apply each case of (12) to Ω_1, assuming $2 \in \mathfrak{S} \cap \mathfrak{F}$, which implies $0 \in \mathfrak{S} \cap \mathfrak{F}$ according to (10) and (11).

The first case is ruled out because $1 \in \mathfrak{S} \cap \mathfrak{F}$ implies that $\nexists j : j >_{RU} 1$ and we know that $2 >_{RU} 1$.

In the second case, $1 \in \mathfrak{S} \setminus \mathfrak{F}$ implies that $\nexists j : (j >_{RU} 1) \wedge (\mathscr{T}_{p(j)} \text{ is } Seq)$, thus node 0 must be a Fallback. With the application of (8), (9), and (12), we then have $\Omega_0 = I_0$, $\Omega_1 = I_1 \cap (R_1 \cup S_1) = I_0 \cap (R_1 \cup S_1)$, and $\Omega_2 = I_2 = I_0 \cap F_1$. From this, we see that $\Omega_1 \cap \Omega_2 = I_0 \cap (R_1 \cup S_1) \cap I_0 \cap F_1 = \emptyset$ because $\{R_1, S_1, F_1\}$ are pairwise disjoint by (3). Additionally, $\Omega_1 \cup \Omega_2 = (I_0 \cap (R_1 \cup S_1)) \cup (I_0 \cap F_1) = I_0 \cap (R_1 \cup S_1 \cup F_1) = I_0 = \Omega_0$.

In the third case, $1 \in \mathfrak{F} \setminus \mathfrak{S}$ implies that $\nexists j : (j >_{RU} 1) \wedge (\mathscr{T}_{p(j)} \text{ is } Fal)$ thus node 0 must be a Sequence. With the application of (7), (9), and (12), we then have $\Omega_0 = I_0$, $\Omega_1 = I_1 \cap (R_1 \cup F_1) = I_0 \cap (R_1 \cup F_1)$, and $\Omega_2 = I_2 = I_0 \cap S_1$. From this, we see that $\Omega_1 \cap \Omega_2 = I_0 \cap (R_1 \cup F_1) \cap I_0 \cap S_1 = \emptyset$ because $\{R_1, S_1, F_1\}$ are pairwise disjoint by (3). Additionally, $\Omega_1 \cup \Omega_2 = (I_0 \cap (R_1 \cup F_1)) \cup (I_0 \cap S_1) = I_0 \cap (R_1 \cup S_1 \cup F_1) = I_0 = \Omega_0$.

The fourth case's proof follows similarly with $\Omega_1 = I_1 \cap R_1$. The proofs for the cases of (12) for Ω_2 are also similar. ■

We will now formally prove that the state's presence in Ω_i is indeed a sufficient condition to conclude that $\mathscr{T}_i$ is being executed.

Theorem 2: Let P be the set of leaf nodes whose operating regions are non-empty:

$$P := \{i \in V | (\Omega_i \neq \emptyset) \wedge (\nexists j \in V : j >_P i)\}. \quad (13)$$

Then, we have $x \in \Omega_i : i \in P \implies \dot{x} = f(x, u_0(x)) = f(x, u_i(x))$ and $\bigcup_{i \in P} \Omega_i = \mathbb{R}^n$.

Proof: We need to show that $x \in \Omega_i : i \in P \implies f(x, u_0(x)) = f(x, u_i(x))$ and that $\{\Omega_i\}_{i \in P}$ cover the state space.

We have that $\Omega_i \subset I_i$ by (12) and from (9) we see that no leaf to the left of u_i can execute. Furthermore, by the construction of (12), either $x \in R_i$, or x is in the success or failure region of a node on a success or failure pathway (respectively), so no leaf to the right of u_i can execute. Thus, we conclude that $x \in \Omega_i : i \in P \implies f(x, u_0(x)) = f(x, u_i(x))$.

Now we need to show that $\{\Omega_i\}_{i \in P}$ cover the state space. From Lemma 3 we have that Ω_i for a set of siblings are pairwise disjoint and cover $\Omega_{p(i)}$. By definition, $I_0 = \mathbb{R}^n$ and since $0 \in \mathfrak{S} \cap \mathfrak{F}$ we have $\Omega_0 = I_0 = \mathbb{R}^n$ by (12). Applying Lemma 3 recursively down the tree we see that for the leaves in P we have that $\{\Omega_i\}_{i \in P}$ are pairwise disjoint and cover $\mathbb{R}^n$, $\bigcup_{i \in P} \Omega_i = \mathbb{R}^n$. ■

Theorem 3: The execution (4) will have a unique Filippov solution (see [11]) for each initial state if, for every pair of neighboring sets with index in P, i.e., sets Ω_i, Ω_j with $i, j \in P$ and $\partial\Omega_i \cup \partial\Omega_j \neq \emptyset$, the sets Ω_i, Ω_j and the vector field

$$X(x) = \begin{cases} f(x, u_i(x)) & \text{if } x \in \Omega_i \\ f\big(x, u_j(x)\big) & \text{else} \end{cases} \quad (14)$$

are such that the following holds with $D_1 = \Omega_i$ and $D_2 = \mathbb{R}^n \setminus D_1$. $S_X = \partial D_i$ is the set where $X(x)$ is discontinuous and S_X is a C^2-manifold. Furthermore, for $i \in \{1, 2\}$, $X_{|\bar{D}_i}$ is continuously differentiable on D_i and $X_{|\bar{D}_1} - X_{|\bar{D}_2}$ is continuously differentiable on S_X. For each $x \in S_X$, either $X_{|\bar{D}_1}$ points into D_2 or $X_{|\bar{D}_2}$ points into D_1.

Proof: A straightforward application of Theorem 1 for every neighboring pair of Ω_i. ■

Sufficient conditions for the existence and uniqueness of BT executions can thus be found using the corresponding results for DDS in Theorem 1.

VI. Convergence Analysis

In this section, we will state the conditions under which a general BT is convergent. The main idea of our convergence theorem is similar to the concept of *prepares* from [12]. Given a BT and its operating regions, the region of attraction of each policy invokes switching between operating regions, thereby inducing a partial order $\leq_f$ of transitions.

The reflexive-transitive reduction of this partial order is a directed acyclical graph (prepares graph), as illustrated in

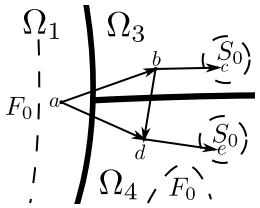

Fig. 3. Prepares graph for the BT in Fig. 2.

Fig. 3 for the kitchen-lamp example in Section IV. The transitions (edges) of this graph are described as follows: (a, b) going to the kitchen and trying to turn on lamp A because it is closer, (a, d) going to the kitchen and trying to turn on lamp B because it is closer, (b, d) trying to turn on lamp B because lamp A did not work, (b, c) successfully turning on lamp A, (d, e) successfully turning on lamp B. Note, the dashed regions in Fig. 3 correspond to the success and failure pathways. Informally speaking, the BT will be convergent if this graph is acyclical and has all its sinks in success regions. We will now formally state the convergence theorem.

Theorem 4: If there exists a subset $L \subseteq P$ and a partial order $\leq_f \subset L^2$ such that the constraint region

$$\Lambda_i := \bigcup_{j \geq_f i} \Omega_j \setminus F_0 \tag{15}$$

is invariant under $f(x, u_i(x))$ for all $i \in L$, and there exists a finite time $\tau_i > 0$, such that if $x(t) \in \Omega_i \setminus S_0$ then $x(t + \tau_i) \notin \Omega_i \setminus S_0$ for all $i \in L$, then there exists a maximum number of transitions $N \in \mathbb{N}$ and a maximum duration $t' > 0$, such that if $x(0) \in \Lambda_i$ for any $i \in L$, then $x(t) \in S_0$ in bounded time $t \leq t'$ within N transitions.

Proof: We have that if $x(t) \in \Omega_i \setminus S_0$ then $x(t+\tau_i) \notin \Omega_i \setminus S_0$. But, Λ_i is invariant under $f(x, u_i(x))$. Thus, if $x(t) \in \Lambda_i$ then $x(t+\tau_i) \in \Omega_j \setminus F_0$ for some $j \geq_f i$, meaning that either $x(t+\tau_i) \in R_0$ or $x(t + \tau_i) \in S_0$. Thus, if $x(0) \in \Lambda_i$ then $x(t) \in S_0$ in bounded time $t \leq t'$ with $t' = \max_{L_0 \subseteq L} \sum_{k \in L_0} \tau_k$ and at most $N = \max_{L_1 \subseteq L} |L_1|$ transitions, such that L_0, L_1 are maximal, totally ordered by $\leq_f$, and $i \leq_f k$ for all $k \in L_0 \cup L_1$. In other words, L_0 and L_1 are the chains of transitions with the largest duration and cardinality, respectively. ∎

We now have a tool to assess the convergence properties of a general BT. The key challenge is thus to design the structure of the BT itself and its controllers to satisfy Theorem 4. An extended version of this letter, with a longer example of the application of this result can be found in [16].

VII. Conclusion

In this letter, we have formulated BTs in continuous-time and shown how they fit the formalism of a DDS and the conditions under which solutions to their execution exist and are unique. To do this, we embedded the order of the BT structure itself into the formulation. These contributions allow the application of the rich literature in hybrid dynamical systems [17]–[19] to BTs in general. Finally, we have provided the conditions under which a general BT will be convergent to a goal.

References

[1] O. Biggar, M. Zamani, and I. Shames, "On modularity in reactive control architectures, with an application to formal verification," 2020, *arXiv:2008.12515*.
[2] D. Isla, "Handling complexity in the halo 2 AI," in *Proc. Game Develop. Conf. (GDC)*, 2005.
[3] O. Biggar, M. Zamani, and I. Shames, "An expressiveness hierarchy of behavior trees and related architectures," *IEEE Trans. Robot. Autom.*, vol. 6, no. 3, pp. 5397–5404, Jul. 2021.
[4] M. Iovino, E. Scukins, J. Styrud, P. Ögren, and C. Smith, "A survey of behavior trees in robotics and AI," 2020, *arXiv:2005.05842*.
[5] P. Ögren, "Increasing modularity of UAV control systems using computer game behavior trees," in *Proc. AIAA Guid. Navig. Control Conf.*, 2012, p. 4458.
[6] A. Marzinotto, M. Colledanchise, C. Smith, and P. Ögren, "Towards a unified behavior trees framework for robot control," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, Hong Kong, May/Jun. 2014, pp. 5420–5427.
[7] M. Colledanchise and P. Ögren, "How behavior trees modularize hybrid control systems and generalize sequential behavior compositions, the subsumption architecture, and decision trees," *IEEE Trans. Robot.*, vol. 33, no. 2, pp. 372–389, Apr. 2017.
[8] A. Klöckner, "The modelica behaviortrees library: Mission planning in continuous-time for unmanned aircraft," in *Proc. 10th Int. Modelica Conf.*, Dec. 2014, pp. 727–736.
[9] C. Paxton, N. D. Ratliff, C. Eppner, and D. Fox, "Representing robot task plans as robust logical-dynamical systems," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Macau, China, Nov. 2019, pp. 5588–5595.
[10] P. Ögren, "Convergence analysis of hybrid control systems in the form of backward chained behavior trees," *IEEE Robot. Autom. Lett.*, vol. 5, no. 4, pp. 6073–6080, Oct. 2020.
[11] J. Cortes, "Discontinuous dynamical systems," *IEEE Control Syst. Mag.*, vol. 28, no. 3, pp. 36–73, Jun. 2008.
[12] R. R. Burridge, A. A. Rizzi, and D. E. Koditschek, "Sequential composition of dynamically dexterous robot behaviors," *Int. J. Robot. Res.*, vol. 18, no. 6, pp. 534–555, 1999.
[13] C. I. Sprague and P. Ögren, "Adding neural network controllers to behavior trees without destroying performance guarantees," 2018, *arXiv:1809.10283*.
[14] D. C. Conner, H. Choset, and A. Rizzi, "Integrated planning and control for convex-bodied nonholonomic systems using local feedback," in *Proc. Robot. Sci. Syst. (RSS)*, Aug. 2006, pp. 57–64.
[15] T. Kuboyama, "Matching and learning in trees," M.S. Thesis, Dept. Adv. Interdiscipl., Stud., Univ. Tokyo, Tokyo, Japan, 2007.
[16] C. I. Sprague and P. Ögren, "Continuous-time behavior trees as discontinuous dynamical systems," 2021, *arXiv:2109.01575*.
[17] M. S. Branicky, "Multiple Lyapunov functions and other analysis tools for switched and hybrid systems," *IEEE Trans. Autom. Control*, vol. 43, no. 4, pp. 475–482, Apr. 1998.
[18] R. A. Decarlo, M. S. Branicky, S. Pettersson, and B. Lennartson, "Perspectives and results on the stability and stabilizability of hybrid systems," *Proc. IEEE*, vol. 88, no. 7, pp. 1069–1082, Jul. 2000.
[19] J. P. Hespanha and A. S. Morse, "Stability of switched systems with average dwell-time," in *Proc. 38th IEEE Conf. Decis. Control*, vol. 3. Phoenix, AZ, USA, 1999, pp. 2655–2660.