How Behavior Trees Modularize Robustness and Safety in Hybrid Systems

Michele Colledanchise and Petter Ögren

***Abstract*— Behavior Trees (BTs) have become a popular framework for designing controllers of in-game opponents in the computer gaming industry. In this paper, we formalize and analyze the reasons behind the success of the BTs using standard tools of robot control theory, focusing on how properties such as robustness and safety are addressed in a modular way. In particular, we show how these key properties can be traced back to the ideas of subsumption and sequential compositions of robot behaviors. Thus BTs can be seen as a recent addition to a long research effort towards increasing modularity, robustness and safety of robot control software. To illustrate the use of BTs, we provide a set of solutions to example problems.**

I. Introduction

Behavior Trees (BTs) were developed within the computer gaming community [1], [2], [3] as a more modular alternative to Finite State Machines (FSMs). Their recursive structure and usability have made them very popular in industry, which in turn has created a growing amount of attention in academia [4], [5], [6], [7], [8], [9], [10], [11].

The main advantage of BTs compared to FSMs is their modularity, as can be seen by the following programming language analogy [2]. In FSMs, the state transitions are encoded in the states themselves, and switching from one state to the other leaves no memory of where the transition was made from, a so-called *one way control transfer*. This is very general and flexible, but actually very similar to the now obsolete *GOTO command*, that was an important part of many early programming languages, e.g., BASIC. In BTs the equivalents of state transitions are governed by calls and return values being passed up and down the tree structure, i.e. *two way control transfers*. This is also flexible, but more similar to the use of *function calls*, that has replaced GOTO in almost all modern programming languages. Using function calls when programming made it much easier to modularize the code, which in turn improved readability and reusability. Thus, BTs exhibit many of the advantages in terms of modularity (including readability and reusability) that was gained when going from GOTO to function calls in the 1980s. Note however, that there are *no* claims that BTs are superior to FSMs from a purely theoretical standpoint. On the contrary, all BTs can most likely be formulated in terms of a FSM, just as most general purpose programing languages are equivalent in the sense of Turing completeness, but still differ in modularity, readability and reusability.

The authors are with the Computer Vision and Active Perception Lab., Centre for Autonomous Systems, School of Computer Science and Communication, KTH - Royal Institute of Technology, SE-100 44 Stockholm, Sweden. e-mail: {miccol|petter}@kth.se

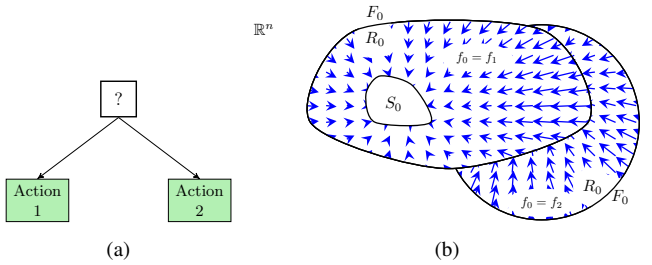

Fig. 1. A minimalist Behavior Tree composition (a) and the corresponding vector field (b). The second subtree increases the robustness of the composition by increasing the combined region of attraction.

BTs were first described in [1], [2], as powerful tools to provide intelligent behaviors for non-player characters in high profile computer games, such as the HALO series. Later work proposed ways of combining BTs with machine learning techniques [4], [5], and making them more flexible in terms of parameter passing [6]. The advantage of BTs as compared to FSMs was also the reason for extending the JADE agent Behavior Model with BTs in [7], and the benefits of using BTs to control complex multi mission UAVs was described in [8]. The modularity and structure of BTs enabled a step towards the formal verification of mission plans in [10], and estimates on execution times of stochastic BTs were provided in [11]. A ROS node implementation of BTs was described in [12] and a Modellica implementation in [13]. Finally, in [9], BTs were used to perform autonomous robotic grasping in the Darpa Autonomous Robotic Manipulation Challenge. In particular, it was shown how BTs enabled the easy composition of primitive actions into complex and robust manipulation programs.

In this paper, we show that the safety and robustness (in terms of large regions of attraction, see Figure 1) of BTs to a large extent relies upon ideas from the robotics community, presented in [14] and [15]. In a seminal paper by Burridge et al., [14] the idea of increasing the region of attraction of a controller by composing a set of controllers was described. By making sure that the asymptotically stable equilibrium of one controller was inside the region of attraction of the next, stability properties of the combination could be proved. This line of thought was later continued in e.g., [15], where additional robustness issues were considered. In another ground breaking paper [16], ideas regarding the more of less parallel execution of controllers of different priority were described. The key idea being that a higher priority controller can subsume or suppress, the lower priority ones,

whenever needed. Both these ideas are fundamental to the BT structure.

The contributions of this paper is that we present a model of BTs that is explicitly based on recursive function calls, allowing us to investigate properties such as robustness and safety. We also provide a theoretical analysis of when these properties are preserved in recursive modular compositions of new BTs and describe how these compositions rely on the ideas described in [14] and [16]. Finally, the fact that BTs are used extensively to design hybrid controllers in the computer gaming industry, motivates the analysis of them from a robotics control perspective.

The outline of this paper is as follows. In Section II we review the classical formulation of BTs . Then, in Section III we give a number of examples of BTs in different areas of application. Section IV-A then contains a new compact function call formulation of BTs. Using the new formulation, we define properties related to robustness and safety for BTs in Section IV-B, and prove how they are carried over to composite BTs in Section IV-C. Finally, the paper is concluded in Section V.

II. Background: Classical formulation of BTs

In this section, we will describe BTs in the classical way. To enable our analysis, we will then, in Section IV, provide a functional model of BTs. The classical description is included for comparison with the functional one.

Following [8], we let a BT be a directed tree, with nodes and edges, using the usual definition of parents and children for neighboring nodes. The node without parents is called the root node, and nodes without children are called leaf nodes. Now, each node of the BT is labeled as belonging to one of the four different types listed in Table I. If the node is not a leaf it can be one of the first two types, *Selector* or *Sequence*, and if it is a leaf it is one of the last two types, *Action* or *Condition*.

Remark 1: Here we only focus on the core concepts of BTs. For a more complete description, including e.g. *Decorator* and *Parallel* nodes, see see [1], [2], [3], [8].

Upon execution of the BT, each time step of the control loop, the root of the BT is *ticked*. This tick is then progressed down the tree according to the types of each node. Once a tick reaches a leaf node (Action or Condition), the node does some computation, possibly affecting some continuous or discrete states/variables of the BT, and then returns either *Success*, *Failure* or *Running*. The return status is then progressed up the tree, back towards the root, according to the types of each node. We will now describe how all the different node types handle the tick and processes the different return statuses.

Selector. Selectors are used to find and execute the first child that does not fail. A Selector will return immediately with a status code success or running when one of its children returns success or running, see Table I and the pseudo code below. The children are ticked in order of importance, from left to right. Figure 2 illustrates a simple BT with one selector and a set of Actions. We will later discuss how this is related to the ideas in [14].

Algorithm 1: Pseudocode of a Selector node with N children

```
for i ← 1 to N do
    childStatus ← Tick(child(i))
    if childStatus = running then
        return running
    else if childStatus = success then
        return success
return failure
```

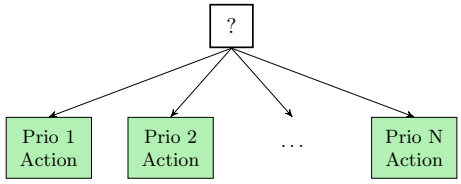

Fig. 2. The Selector ticks its children in order until one returns Success or Running. Selectors are denoted by a white square with a question mark and Actions are denoted by a green square.

Sequence. Sequences are used to find and execute the first child that has not yet succeeded. A Sequence will return immediately with a status code failure or running when one of its children returns failure or running, see Table I and the pseudo code below. The children are ticked in order, from left to right. Figure 3 illustrates a simple BT with one Sequence and a set of Actions. We will later discuss how this is related to the ideas in [16].

Algorithm 2: Pseudocode of a Sequence node with N children

```
for i ← 1 to N do
    childStatus ← Tick(child(i))
    if childStatus = running then
        return running
    else if childStatus = failure then
        return failure
return success
```

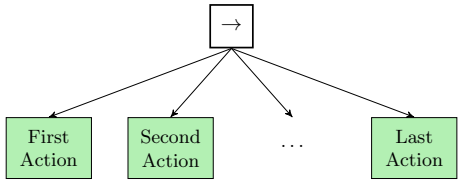

Fig. 3. The Sequence ticks its children in order until one returns failure or running.

Action. An Action node performs an action, and returns

TABLE I. The six node types of a BT.

Node type	Succeeds	Fails	Running
Selector	If one child succeeds	If all children fail	If one child returns running
Sequence	If all children succeeds	If one child fails	If one child returns running
Action	Upon completion	When impossible to complete	During completion
Condition	If true	If false	Never

Success if the action is completed, Failure if it can not be completed and Running if completion is under way.

Condition. A Condition node determines if a condition C has been met. Conditions are technically a subset of the Actions, but are given a separate category and graphical symbol to improve readability of the BT and emphasize the fact that they never return running and do not change any internal states/variables of the BT. Examples of Conditions can be found in Figure 4 below.

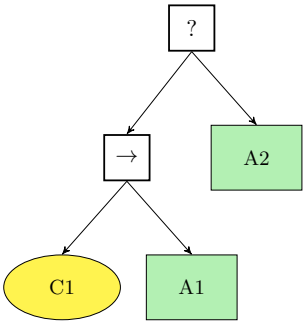

Fig. 4. A Selector, a Sequence, a Condition (yellow) and two Actions (green). Action A1 is only performed when Condition C1 returns Success and Action A2 is only performed if C1 or A1 returns Failure.

III. Examples of BTs

To see how BTs are used to provide modularity, robustness and safety, we will review a set of examples. For the purpose of illustration, they are all quite small. Note however that the application reported in [9], Figure 6, was part of the DARPA Autonomous Robotic ManipulationSoftware track (ARM-S) challenge. Thus BTs have been successfully used in *real robotic applications*, with the explicit motivation of being "an architecture supporting easy composition of primitive behaviors into complex and robust manipulation programs", [9]. This paper investigates the robustness and safety of such compositions.

As noted above, BTs were developed in the gaming industry to provide hybrid controllers for in game opponents. In [17] the problem of machine learning was investigated for BTs controlling game characters doing hand to hand combat. One of the BTs used is depicted in Figure 5. Note how the combination of Sequence-Condition-Action from the left of Figure 4 is used repeatedly to determine when to use the different controllers. When the enemy is not in sight, the character rests.

A completely different BT is shown in Figure 6. This is a sub tree of the BT doing grasping in [9]. As can be seen,

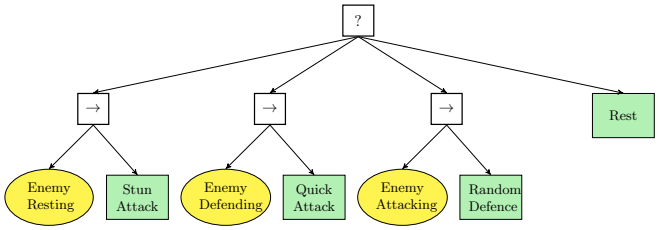

Fig. 5. A BT for doing hand to hand combat in a computer game.

the robot tries to execute the sequence of finding the table, detecting the objects, and verifying the pose of the hammer. If any of these actions return failure, the robot will move to improve/switch the perspective, until progress in the task sequence is once again established. One can assume that *Switch perspective* always returns Running, which implies that the complete BT only returns success when all three actions in the main sequence have succeeded.

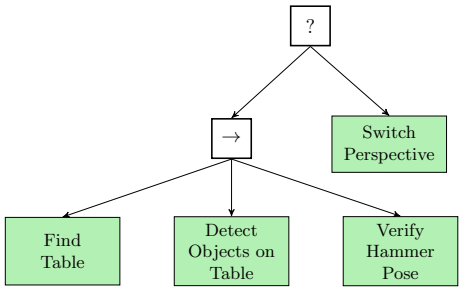

Fig. 6. If the key actions in the first sequence fail, the robot moves to improve the perspective, until progress in the sequence resumes.

The example in Figure 7 deals with autonomous driving. It is meant to be a BT version of the FSM used in [18]. The first, default, action is basic driving in a lane. This corresponds to invoking the car controller with the following constraints: no passing, no reversing, keep large safety margins. If no progress can be achieved with these constraints, implying that the current lane is blocked, the next action invokes the same controller while allowing passing, but reducing the allowed speed. If again, no progress can be made, perhaps because the vehicle in front is too close, the constraint of reversing is removed while the allowed speed is reduced further. If the lack of progress remains, the safety margins are stepwise reduced along with the allowed speed. Until finally, extremely low speed off road driving is applied to pass the obstacle.

The example in Figure 8 also deals with autonomous driving. Here the special cases presented by parking lots,

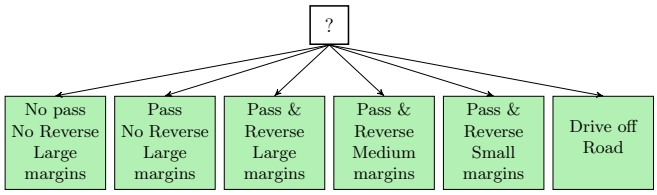

Fig. 7. The BT of an autonomous car, loosely following [18].

intersections, and needed u-turns are first covered, as described in [19]. If non of those are needed, the vehicle tries to perform normal lane following. Finally, if lane following fails (i.e. results in a stop due to vehicles blocking the road) an overtaking action is performed.

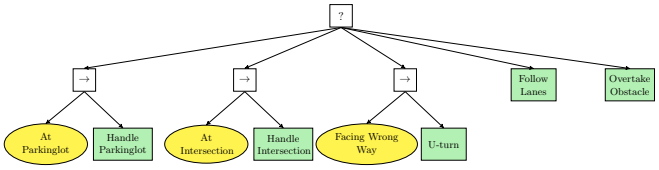

Fig. 8. The BT of an autonomous car, loosely following [19].

The final example is a UAV controller, where the safety critical part of the control system has been moved into a separate module and connected in a sequence with the rest, see Figure 9. Thus, the rest is executed only when the *Guarantee altitude above 1000 ft* returns success. In this way, the complex mission execution part can be updated and extended, without risk of introducing bugs in the safety critical part.

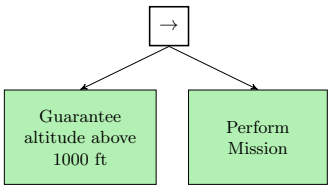

Fig. 9. A UAV control BT, where the safety critical part of the controller has been isolated from the complex mission execution part.

Remark 2: Note that all examples above could be captured using a FSM instead of a BT. As noted above, the advantages of BTs lie in the modularity of the switching, making compositions easier and improving e.g. readability and reusability. We believe that these advantages, in terms of software complexity, are as important when developing robotics software, as it is when developing computer games.

IV. Main Results

In this section, we will first propose a functional model of BTs, we then define properties regarding safety, robustness and efficiency of BTs, and finally show how these properties extend across BT compositions in a modular way.

A. A Functional Model of BTs

In this section we define a more formal, functional version of the BTs described above. The *tick* is now replaced by recursive function calls, incorporating both the return status, and the dynamics of the control system. These definitions enable us to describe and prove properties of the BTs.

Definition 1 (Behavior Tree (BT)): A BT is a three-tuple

$$\mathcal{T}_i = \{f_i, r_i, \Delta t\}, \tag{1}$$

where $i \in \mathbb{N}$ is the index of the tree, $f_i : \mathbb{R}^n \to \mathbb{R}^n$ is the right hand side of an ordinary difference equation, Δt is a time step and $r_i : \mathbb{R}^n \to \{R, S, F\}$ is the return status, that can be equal to either *Running* (R), *Success* (S), or *Failure* (F). The return status r_i will be used when recursively combining BTs, as explained below.

Definition 2 (Executing a BT): The execution of a BT $\mathcal{T}_i$ is a standard ordinary difference equation

$$x_{k+t}(t_{k+1}) = f_i(x_k(t_k)), \tag{2}$$
$$t_{k+1} = t_k + \Delta t. \tag{3}$$

From now on we will assume that all BTs evolve in the same continuous space $\mathbb{R}^n$ using the same time step Δt_i.

Remark 3: It is often the case, that different BTs, controlling different vehicle subsystems evolving in different state spaces, need to be combined into a single BT. Such cases can be fit in the assumption above by letting all systems evolve in a larger state space, that is the cartesian product of the smaller state spaces.

Definition 3: The three regions $R_i, S_i, F_i \subset \mathbb{R}^n$ of a BT $\mathcal{T}_i$ are defined as follows

$$R_i = \{x : r_i(x) = R\} \tag{4}$$
$$S_i = \{x : r_i(x) = S\} \tag{5}$$
$$F_i = \{x : r_i(x) = F\} \tag{6}$$

and denoted Running/Activation region (R_i), Success region (S_i) and Failure region (F_i).

A behavior tree that never returns running, is called a Condition. For those, success/failure are often interpreted as true/false:

Definition 4 (Condition): A Condition is a BT $\mathcal{T}_i$ with $R_i = \varnothing$.

BTs that satisfy Definition 1 directly, without calling other subtrees, see below, is called Actions.

Definition 5 (Action): An Action is a BT $\mathcal{T}_i$ that has no subtrees.

Definition 6 (Sequence compositions of BTs): Two or more BTs can be composed into a more complex BT using a Sequence operator,

$$\mathcal{T}_0 = \text{Sequence}(\mathcal{T}_1, \mathcal{T}_2).$$

Then r_0, f_0 are defined as follows

$$\begin{aligned} &\text{If } x_k \in S_1 && && (7)\\ &\quad r_0(x_k) &&= r_2(x_k) && (8)\\ &\quad f_0(x_k) &&= f_2(x_k) && (9)\\ &\text{else} && && \\ &\quad r_0(x_k) &&= r_1(x_k) && (10)\\ &\quad f_0(x_k) &&= f_1(x_k). && (11)\end{aligned}$$

$\mathcal{T}_1$ and $\mathcal{T}_2$ are called children of $\mathcal{T}_0$. Note that when executing the new BT, $\mathcal{T}_0$ first keeps executing its first child $\mathcal{T}_1$ as long as it returns Running or Failure. The second child is executed only when the first returns Success, and $\mathcal{T}_0$ returns Success only when all children have succeeded, hence the name Sequence. For notational convenience, we write

$$\text{Sequence}(\mathcal{T}_1, \text{Sequence}(\mathcal{T}_2, \mathcal{T}_3)) = \text{Sequence}(\mathcal{T}_1, \mathcal{T}_2, \mathcal{T}_3), \tag{12}$$

and similarly for arbitrarily long compositions.

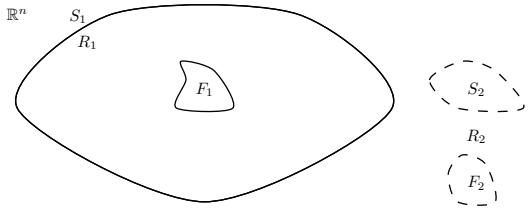

Fig. 10. The sets S_1, F_1, R_1 (solid boundaries) and S_2, F_2, R_2 (dashed boundaries) of Example 1 and Lemma 2.

Example 1: To illustrate how safety can be improved using a Sequence composition, we once again consider the BT in Figure 9. The sets S_i, F_i, R_i are shown in Figure 10. As $\mathcal{T}_1$ is *Guarrantee altitude above 1000 ft*, its failure region F_1 is a small part of the state space (corresponding to a crash) surrounded by the running region R_1 that is supposed to move the UAV away from the ground, guaranteeing a minimum altitude of 1000 ft. The success region S_1 is large, every state sufficiently distant from F_1. The BT that performs the mission, $\mathcal{T}_2$, has a smaller success region S_2, surrounded by a very large running region R_2, containing a small failure region F_2. The function f_0 governed by Equations (9) and (11) and is depicted together with the vector field $(f_0(x)-x)$ in Figure 11.

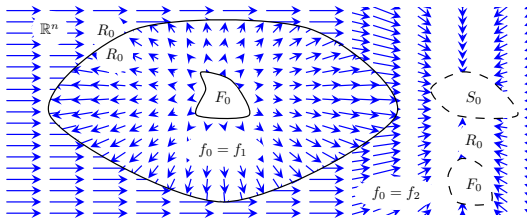

Fig. 11. The sets S_0, F_0, R_0 and the vector field $(f_0(x)-x)$ of Example 1 and Lemma 2.

The discussion above is formalized in Lemma 2 below.

Remark 4: The definition above corresponds to so-called memoryless Sequences. Most BT implementations also include a Sequence with memory, where a subtree that returned Succeed is never executed again, see Remark 1.

Remark 5: The issue of undesired chattering, i.e., switching back and fourth between different sub-controllers, is always an important concern when designing switched control systems. BTs are no exception. As is suggested by the left part of Figure 11, chattering can be a problem when vector fields meet at a switching surface.

Definition 7 (Selector compositions of BTs): Two or more BTs can be composed into a more complex BT using a Selector operator,

$$\mathcal{T}_0 = \text{Selector}(\mathcal{T}_1, \mathcal{T}_2).$$

Then r_0, f_0 are defined as follows

$$\begin{aligned} &\text{If } x_k \in \mathcal{F}_1 && && (13)\\ &\quad r_0(x_k) &&= r_2(x_k) && (14)\\ &\quad f_0(x_k) &&= f_2(x_k) && (15)\\ &\text{else} && && \\ &\quad r_0(x_k) &&= r_1(x_k) && (16)\\ &\quad f_0(x_k) &&= f_1(x_k). && (17)\end{aligned}$$

Note that when executing the new BT, $\mathcal{T}_0$ first keeps executing its first child $\mathcal{T}_1$ as long as it returns Running or Success. The second child is executed only when the first returns Failure, and $\mathcal{T}_0$ returns Failure only when all children have tried, but failed, hence the name Selector.

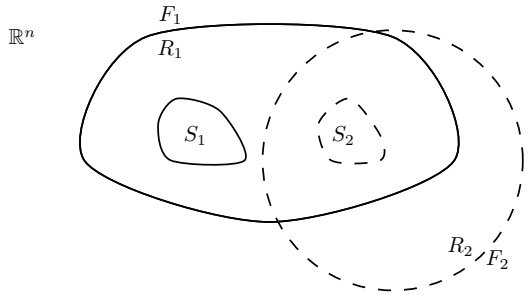

Fig. 12. The sets S_1, F_1, R_1 (solid boundaries) and S_2, F_2, R_2 (dashed boundaries) of Example 2 and Lemma 2.

Example 2: To illustrate how robustness can be improved using a Selector composition, we consider a general BT with two subtrees, see Figure 1(a), that might correspond to the two first subtrees of the Autonomous driving BT in in Figure 7. The sets S_i, F_i, R_i are shown in Figure 12. $\mathcal{T}_1$ is the main BT, expected to be executing most of the time and get us to the desired part of the state space. However, there are some situations that $\mathcal{T}_1$ cannot handle, for example when overtaking is needed to pass a blocking vehicle. Instead of making $\mathcal{T}_1$ more complex, we combine it with another BT, $\mathcal{T}_2$, that can handle the situation, i.e. do overtaking, and move the state back to the part of the state space that $\mathcal{T}_1$ can handle, this corresponds to $S_2 \subset R_1$. The sets S_0, F_0, R_0 and f_0 of the combined BT are shown in Figure 13, together with the

vector field $f_0(x) - x$. As can be seen, the combined BT can now move a larger set of initial conditions to the desired region $S_0 = S_1$.

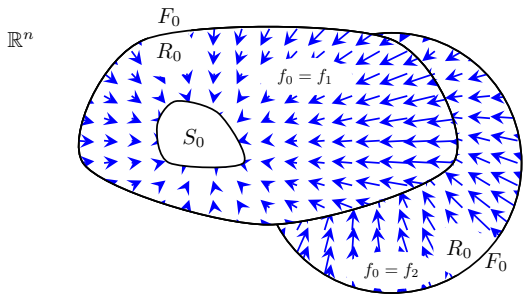

Fig. 13. The sets S_0, F_0, R_0 and the vector field $(f_0(x)-x)$ of Example 2 and Lemma 3.

The discussion above is formalized in Lemma 3 below.

B. Definitions of Safety, Efficiency and Robustness of BTs

Many control problems, in particular in robotics, can be formulated in terms of achieving a given goal configuration in a way that is safe, time efficient, and robust with respect to the initial configuration. Given a configuration space, safety corresponds to avoiding parts of the space, time efficiency corresponds to reaching another part of the space in time, and robustness corresponds to achieving both of the above from a large set of initial positions.

Using the definitions below, we say that a BT is *safe* if it satisfies Definition 8, *efficient* if it satisfies Definition 9 with a small τ and *robust* if it satisfies Definition 9 with a large region of attraction R'.

Definition 8 (Safe): A BT is Safe, with respect to the obstacle region $O \subset \mathbb{R}^n$, and the initialization region $I \subset R$, if for all starting points $x(0) \in I$, $x(t) \notin O$, for all $t \geq 0$.

Definition 9 (Finite Time Successful): A BT is Finite Time Successful with region of attraction R', if for all starting points $x(0) \in R' \subset R$, there is a time τ such that $x(\tau') \in S$ for some $\tau' \leq \tau$ and $x(t) \in R'$ for all $t \in [0, \tau')$.

As noted in the following Lemma, exponential stability implies Finite Time Success, given the right choices of the sets S, F, R.

Lemma 1 (Exponential stability and FTS): A BT for which x_s is a globally exponentially stable equilibrium of the execution (2), and $S \supset \{x : \|x - x_s\| \leq \epsilon\}$, $\epsilon > 0$, $F = \varnothing$, $R = \mathbb{R}^n \setminus S$, is Finite Time Successful.

Proof: Global exponential stability implies that there exists $a > 0$ such that $\|x(k) - x_s\| \leq e^{-ak}$ for all k. Then, for each ϵ there is a time τ such that $\|x(k) - x_s\| \leq e^{-a\tau} < \epsilon$, which implies that there is a $\tau' < \tau$ such that $x(\tau') \in S$ and the BT is Finite Time Successful. ■

In order to make statements about the safety of composite BTs we also need the following definition.

Definition 10 (Safeguarding): A BT is Safeguarding, with respect to the step length d, the obstacle region $O \subset \mathbb{R}^n$, and the initialization region $I \subset R$, if it is safe, and finite time successful with region of attraction $R' \supset I$ and a success region S such that I surrounds S in the following sense:

$$\{x \in \mathbb{R}^n : \inf_{s \in S_1} \|x - s\| \leq d\} \subset I. \tag{18}$$

We are now ready to look at how these properties extend across compositions of BTs.

C. Safety, Efficiency and Robustness of Composite BTs

In this section we use standard tools from control theory to show how properties of composite BTs can be derived from their subtrees. We believe that this functional modularity provides the reusability and flexibility that are the key reasons behind the success of BTs in the computer gaming industry.

In the first result, we show that the modularity of the Safety property using the Sequence composition, builds upon the ideas of subsumption, originally presented in [16].

Lemma 2 (Safety of Sequence Compositions): If $\mathcal{T}_1$ is safeguarding, with respect to the obstacle O_1 initial region I_1, and margin d, and $\mathcal{T}_2$ is an arbitrary BT with $\max_x \|x - f_2(x)\| < d$, then the composition $\mathcal{T}_0 = \text{Sequence}(\mathcal{T}_1, \mathcal{T}_2)$ is Safe with respect to O_1 and I_1.

Proof: $\mathcal{T}_1$ is safeguarding, which implies that $\mathcal{T}_1$ is safe and thus any trajectory starting in I_1 will stay out of O_1 as long as $\mathcal{T}_1$ is executing. But if the trajectory reaches S_1, $\mathcal{T}_2$ will execute until the trajectory leaves S_1. We must now show that the trajectory cannot reach O_1 without first entering I_1. But any trajectory leaving S_1 must immediately enter I_1, as the first state outside S_1 must lie in the set $\{x \in \mathbb{R}^n : \inf_{s \in S_1} \|x - s\| \leq d\} \subset I_1$ due to the fact that for $\mathcal{T}_2$, $\|x(k) - x(k+1)\| = \|x(k) - f_2(x(k))\| < d$. ■

This Lemma is also illustrated in Example 1 and Figures 10 and 11 above.

The second result regarding robustness, is a variation on the elegant funnel argument of Burridge et al [14]. There, they showed how the region of attraction could be extended using a family of controllers, as long as the asymptotically stable equilibrium of each controller was either the goal state, or inside the region of attraction of another controller, positioned earlier in the sequence.

We will now describe the construction of Burridge et al [14] in some detail, and then see how this concept is captures in the BT framework. Given a family of controllers $U = \{\Phi_i\}$, we say that Φ_i *prepares* Φ_j if the goal $G(\Phi_i)$ is inside the domain $D(\Phi_j)$. Assume the overall goal is located at $G(\Phi_1)$. A set of execution regions $C(\Phi_i)$ for each controller was then calculated according to the following scheme:

1) Let the queue contain Φ_1. Let $C(\Phi_1) = D(\Phi_1)$, $N = 1$, $D_1 = D(\Phi_1)$.
2) Remove the first element of the queue and append all controllers that *prepare* it to the back of the queue.
3) Remove all elements in the queue that already has a defined $C(\Phi_i)$.
4) Let Φ_j be the first element in the queue. Let $C(\Phi_j) = D(\Phi_j) \setminus D_N$, $D_{N+1} = D_N \cup D(\Phi_j)$ and $N \leftarrow N + 1$.
5) Repeat steps 2,3 and 4 until the queue is empty.

The combined controller is then executed by finding j such that $x \in C(\Phi_j)$ and then invoking controller Φ_j.

Looking at the design of the Selector operator in BTs, it turns out that it does exactly the job of the Burridge algorithm above, as long as the subtrees of the Selector are ordered in the same fashion as the queue above. We formalize this in the following Lemma.

Lemma 3: (Robustness and Efficiency of Selector Compositions) If $\mathcal{T}_1, \mathcal{T}_2$ are Finite Time Successful, with $S_2 \subset R_1'$, then $\mathcal{T}_0$ = Selector$(\mathcal{T}_1, \mathcal{T}_2)$ is Finite Time Successful with $\tau_0 = \tau_1 + \tau_2$, $R_0' = R_1' \cup R_2'$ and $S_0 = S_1$.

Proof: First we consider the case when $x(0) \in R_1'$. Then, as $\mathcal{T}_1$ is FTS, the state will reach S_1 before $\tau_1 < \tau_0$, without leaving R_1'. If $x(0) \in R_2' \setminus R_1'$, $\mathcal{T}_2$ will execute, and the state will progress towards S_2. But as $S_2 \subset R_1'$, $x(k_1) \in R_1'$ at some time $k_1 < \tau_2$. Then, we have the case above, reaching $x(k_2) \in S_1$ in a total time of $k_2 < \tau_1 + k_1 < \tau_1 + \tau_2$. ■

This Lemma is also illustrated in Example 2 and Figures 12 and 13 above.

The efficiency of some compositions can be computed using Lemma 3 above. But in other cases, efficiency of the combined controllers is reduced significantly by chattering, as noted in Remark 5. Inspired by [20] the following result can give an indication of when chattering is to be expected.

Lemma 4: Given a composition $\mathcal{T}_0$ = Sequence$(\mathcal{T}_1, \mathcal{T}_2)$, where f_i depend on Δt are such that $\|f_i(x) - x\| \to 0$ when $\Delta t \to 0$. Let $s : \mathbb{R}^n \to \mathbb{R}$ be such that $s(x) = 0$ if $x \in \delta S_1 \cap R_2$, $s(x) < 0$ if $x \in$ interior$(S_1) \cap R_2$, $s(x) > 0$ if $x \in$ interior$(\mathbb{R}^n \setminus S_1) \cap R_2$, and let

$$\lambda_i(x) = (\frac{\partial s}{\partial x})^T (f_i(x) - x).$$

Then, $x \in \delta S_1$ is chatter free for small enough δt, if $\lambda_1(x) < 0$ or $\lambda_2(x) > 0$.

Proof: When condition holds, the vector field is pointing outwards on at least one side of the switching boundary. ■

Note that this condition is not satisfied on the left hand side of Figure11. This concludes our analysis of BT compositions.

V. Conclusions

In this paper, we have provided a theoretical description of how properties such as robustness and safety are preserved in modular compositions of BTs. We have also provided a new function call formulation of BTs underlying the analysis, and discussed how BT compositions build upon the earlier ideas of subsumption and sequential composition of robot behaviors. We believe that the strength of BTs lie in their modularity, and that BTs can complement FSM in robotic software development, much like one programming language can complement another.

Acknowledgements

The authors thank Professor Magnus Egerstedt for his valuable input into this paper. This work has been supported by the European Union FP7 Project Reconfig (FP7-ICT-600825), the authors gratefully acknowledge the support.

References

[1] D. Isla, "Handling Complexity in the Halo 2 AI," in *Game Developers Conference*, 2005.

[2] A. Champandard, "Understanding Behavior Trees," *AiGameDev. com*, vol. 6, 2007.

[3] D. Isla, "Halo 3-building a Better Battle," in *Game Developers Conference*, 2008.

[4] C. Lim, R. Baumgarten, and S. Colton, "Evolving Behaviour Trees for the Commercial Game DEFCON," *Applications of Evolutionary Computation*, pp. 100–110, 2010.

[5] D. Perez, M. Nicolau, M. O'Neill, and A. Brabazon, "Evolving Behaviour Trees for the Mario AI Competition Using Grammatical Evolution," *Applications of Evolutionary Computation*, 2011.

[6] A. Shoulson, F. M. Garcia, M. Jones, R. Mead, and N. I. Badler, "Parameterizing Behavior Trees," in *Motion in Games*. Springer, 2011.

[7] I. Bojic, T. Lipic, M. Kusek, and G. Jezic, "Extending the JADE Agent Behaviour Model with JBehaviourtrees Framework," in *Agent and Multi-Agent Systems: Technologies and Applications*. Springer, 2011, pp. 159–168.

[8] P. Ögren, "Increasing Modularity of UAV Control Systems using Computer Game Behavior Trees," in *AIAA Guidance, Navigation and Control Conference, Minneapolis, MN*, 2012.

[9] J. A. D. Bagnell, F. Cavalcanti, L. Cui, T. Galluzzo, M. Hebert, M. Kazemi, M. Klingensmith, J. Libby, T. Y. Liu, N. Pollard, M. Pivtoraiko, J.-S. Valois, and R. Zhu, "An Integrated System for Autonomous Robotics Manipulation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, October 2012, pp. 2955–2962.

[10] A. Klökner, "Interfacing Behavior Trees with the World Using Description Logic," in *AIAA conference on Guidance, Navigation and Control, Boston*, 2013.

[11] M. Colledanchise, A. Marzinotto, and P. Ögren, "Performance Analysis of Stochastic Behavior Trees," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, June 2014.

[12] A. Marzinotto, M. Colledanchise, C. Smith, and P. Ögren, "Towards a Unified Behavior Trees Framework for Robot Control," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, June 2014.

[13] A. Klöckner, F. van der Linden, and D. Zimmer, "The Modelica BehaviorTrees Library: Mission planning in continuous-time for unmanned aircraft," in *Proceedings of the 10th International Modelica Conference*, 2014.

[14] R. R. Burridge, A. A. Rizzi, and D. E. Koditschek, "Sequential Composition of Dynamically Dexterous Robot Behaviors," *The International Journal of Robotics Research*, vol. 18, no. 6, pp. 534–555, 1999.

[15] J. Le Ny and G. J. Pappas, "Sequential Composition of Robust Controller Specifications," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012, pp. 5190–5195.

[16] R. Brooks, "A Robust Layered Control System for a Mobile Robot," *Robotics and Automation, IEEE Journal of*, vol. 2, no. 1, pp. 14–23, 1986.

[17] L. Pena, S. Ossowski, J. M. Pena, and S. M. Lucas, "Learning and Evolving Combat Game Controllers," in *Computational Intelligence and Games (CIG), 2012 IEEE Conference on*. IEEE, 2012, pp. 195–202.

[18] T. Wongpiromsarn and R. M. Murray, "Distributed Mission and Contingency Management for the DARPA Urban Challenge," in *International Workshop on Intelligent Vehicle Control Systems (IVCS)*, 2008.

[19] M. Powers, D. Wooden, M. Egerstedt, H. Christensen, and T. Balch, "The Sting Racing Team's Entry to the Urban Challenge," in *Experience from the DARPA Urban Challenge*. Springer, 2012, pp. 43–65.

[20] A. Filippov and F. Arscott, *Differential Equations with Discontinuous Righthand Sides: Control Systems*, ser. Mathematics and its Applications. Kluwer Academic Publishers, 1988. [Online]. Available: http://books.google.se/books?id=KBDyZSwpQpQC