2019 IEEE 58th Conference on Decision and Control (CDC)
Palais des Congrès et des Expositions Nice Acropolis
Nice, France, December 11-13, 2019

Automatic Safe Behaviour Tree Synthesis for Autonomous Agents

Tadewos G. Tadewos, Laya Shamgah, and Ali Karimoddini

Abstract— **This paper proposes a systematic approach for automatically generating a safe behaviour tree (BT) coordinator to guide an autonomous vehicle to satisfy the goal of a mission without violating the mission safety requirements. The autonomous agent is assumed to have different capabilities to execute multiple actions and operate in a dynamic environment with moving obstacles. For this purpose, we develop a hierarchical and modular synthesis technique that generates safe and reactive Behaviour Trees (BTs) composed of verified actions. At a high level, the synthesized BTs generate a sequence of actions to meet the goals of the mission. Then, the safety issues for the generated BTs are addressed both at the action level by using dynamic differential logic (DL) and globally at the task (BTs) level by forcing a sequence of actions to respect the safety requirement of the successor's action. Adopting the compositional property of the DL allows us to infer the safety of a system from validating its components. Therefore, taking advantages of DL's compostitional property and BT's modular reactivity property, the proposed algorithm synthesizes a safe and correct sequence of actions that can meet the goal and safety requirements of a mission. The details of the developed algorithms are illustrated through an example, verifying the effectiveness of the proposed approach.**

I. Introduction

With advances in technologies, nowadays robots are equipped with advanced sensing, maneuvering, and computing capabilities that enable tasking the robots to accomplish a sophisticated and increasingly complex mission in a dynamic environment [1], [2]. Different approaches exist in the literature for tasking multi-agent systems including, but not limited to, formal specification-guided tasking [3], [4], event-based supervisory control [5]–[7], and mixed-integer linear programming (MILP) [8]. A less computationally expensive way to handle the increasing complexity of tasking multi-agent systems is to employ behaviour trees (BTs) with inherently hierarchical and reactive structures for high-level controller synthesis [9], [10]. However, as the complexity of assigned tasks increases, safety becomes a serious issue, i.e., how a robot can accomplish a task without violating a safety constraint (like avoiding a collision or a restricted zone) and simultaneously meeting the mission objectives.

In [11], a correct-by-construction BT synthesis technique has been introduced, which mainly focuses on generating a BT indirectly from fragmented (subset) LTL (Linear Temporal Logic) specifications. Thus, safe BT generation is limited to specifications expressible by fragmented LTL. In [12], a model checker is used to verify system-level fault tolerance property of a system by directly converting BT models into a suitable syntax acceptable by a model checker for safety analysis. In [13], a similar attempt has been persuade to convert a BT into a process algebra model followed by model checking techniques to validate an LTL specification. In both cases, similar to all model checkers, the techniques are vulnerable to the state explosion problem. In [14], a model checker based slicing, a reduction method that removes inactive parts of a program automatically, method is utilized to formally verify a BT. Even though, this technique can handle a larger model, still the state explosion problem exists. In [15], Dynamic Logic (DL) in conjunction with Satisfiability Modulo Theories (SMT) is used to generate a safe controller from Counter Linear Temporal Logics over a Constraint System (CLTLB(D)) specification. However, the control strategy for this method has to be generated off-line, and thus, an agent utilizing this algorithm cannot autonomously alter the high-level plan to adapt to an emergent scenario. In addition, the target specification is limited to CLTLB(D).

To address the problem of safely in an autonomous system, this paper develops a hierarchical structure where planing is done at the high level and safety is guaranteed at the low level. The high-level planning is done by synthesizing a BT to satisfy the mission goal, while the safety of the BT is guaranteed by verifying the low-level controllers for individual actions using dynamic differential logic (DL) and restricting the composition of actions in such a way that the safety constraint of the successor action is respected by the predecessor action. In general, it is not possible to guarantee the safety of a system from the safety of its components. However, dynamic differential logic (DL) [16], a formal method we use to verify the safety of each action, has an interesting compositional verification property where by only verifying the components and the interactions among the components, the entire BT can be proved to be safe.

The rest of the paper is organized as follows. We provide the necessary preliminaries in Section II. In Section III, the problem of synthesizing safe BT is formulated. Section IV describes the proposed safe BT synthesis mechanism in detail. Section V identifies and verifies safe actions. Section VI illustrates the developed technique using a search and delivery case study. Finally, Section VII concludes the paper.

II. Background

This section provides the description of BT nodes and DL operations.

The authors are with the Department of Electrical and Computer Engineering, North Carolina Agricultural and Technical State University, Greensboro, NC 27411 USA.

Corresponding author: A. Karimoddini. Address: 1601 East Market Street, Department of Electrical and Computer Engineering North Carolina A&T State University Greensboro, NC, US 27411. Email: akarimod@ncat.edu (Tel: +13362853313).

A. Behaviour Tree (BT)

Behavior Tree (BT) is an effective tool to capture the decision making mechanism for an autonomous vehicle. As the name implies, the structure of a BT is based on a tree that can be represented with a directed acyclic graph (DAG) to demonstrate control flows from top to bottom (parent to child) among different types of nodes. At the top of the tree, a *root* node exists that provides the activation clk for all other nodes. In addition to a *root* node, a BT may contain *leaf* and *composite* nodes.

Leaf nodes are terminal nodes that could act as a sensing unit (*condition* nodes) or as a computing/actuation unit (*action* nodes). A *condition* node checks the state of the robot or the environment and return *success* only if the condition is true. An action node performs an operation that modifies/change the state of the robots or the environment. Similarly, the action node returns success only if the operation is completed. Fig. 1.b, shows activation of action A_1 if condition C_1 is true.

Composite nodes provide the capability to compose multiple child nodes under a single parent. A *sequence* node composes actions or sub-trees in an ordered fashion, where activation is passed from one child to the next only if the current node is completed with success. Otherwise, a failure status is returned by the *sequence* node. A *selector* node composes actions or sub-trees with priority where activation of the next child is possible if the current node returns a *failure* status. A selector node return *success* if only one child node succeeds, otherwise it returns *failure*. A *parallel* node provides the capability to execute actions/sub-trees simultaneously. The success of a *parallel* node is determined by a natural number N, which specifies how many children are needed to succeed for the node to return *success*. If N number of children succeed, then the node returns *success*, otherwise it returns *failure*. Figures 1.a, 1.c, and 1.d show the graphical representation of *sequence*, *selector* and *parallel* nodes, respectively.

Generally, the execution of a BT is initiated by the root node which sends a tick (enabling signal) with a certain frequency to its children. Then, the enabled child activates another child or returns its execution status as *running*, *failure*, or *success* to its immediate parent. In this way, the actions are executed from the bottom left of the BT, returning success/failure to their parents.

By the proper combination of leaf nodes (actions and condition nodes), and composite nodes (*sequence, selector or parallel* nodes), a complex BT structure can be constructed that is both modular and reactive and can effectively meet the goal of a mission.

B. Dynamic Differential Logic (DL)

Dynamic differential logic (DL) models and formally verifies a hybrid system by using a textual representation known as Hybrid Program (HP) that provides flexibility with an inherent compositional semantics. The syntax and semantics of HP and DL are defined as follows:

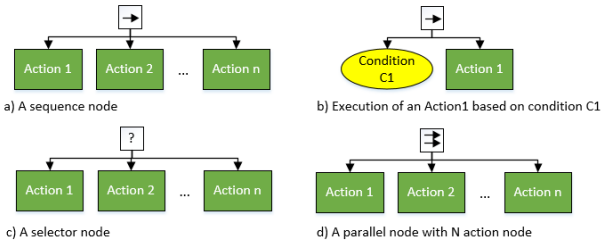

Fig. 1. Building blocks of Behavior Trees

Definition 1: Hybrid Programs (HP):
$\alpha, \beta \mid x_i := \theta_i \mid x_i := * \mid x_1' := \theta_1, \cdots, x_n' := \theta_n \ \&\chi \mid ?\psi \mid \alpha \cup \beta \mid \alpha; \beta \mid \alpha^*$
where α, β are HPs and ψ is a DL formula. Assignments could be as simple as value updates like $x := \theta_i$ which assigns the value θ_i to the state variable x or $x := *$ that assigns a random value $*$ to x. A differential equation update is more complicated, $x_1' := \theta_1, \cdots, x_n' := \theta_n \ \&\chi$ where the set of ordinary differential equations (ODE) are evolved for any duration of time within the evolution domain χ. A tests $?\psi$ evaluates the boolean value of ψ without affecting the state. The nondeterministic choice $\alpha \cup \beta$ selects α or β randomly. The sequential composition $\alpha; \beta$ executes α followed by β. Finally, the repetition α^* execute α a finite number of times.

Definition 2: Dynamic Logic (DL) formula:
$\phi, \psi \mid \neg\phi \mid \psi \wedge \psi \mid \exists x \phi \mid [\alpha]\phi \mid \langle\alpha\rangle \phi$
where the boxed modality $[\alpha]\phi$ always guarantees the satisfaction of ϕ whenever the HP α is executed while the angle modality $\langle\alpha\rangle \psi$ guarantees the satisfaction of ϕ after executing the HP α at least once. In addition, logical operators $(\neg, \wedge, \vee)$, existential $(\exists)$, and universal $(\forall)$ quantifiers are allowed.

DL verification is based on a compositional proof calculus where the satisfaction of a specification is guaranteed by individually verifying each action (component) and inferring system-level safety (proving properties of its parts). The calculus decomposes the system level DL formula $[\alpha]\psi$ symbolically into an equivalent formulas, e.g., $[\alpha_1]\phi_1 \wedge [\alpha_2]\phi_2 \wedge \cdots$ about subsystems α_i of α and sub-properties ϕ_i of ϕ. With this method, $[\alpha]\phi$ can be simply verified by proving the individual sub-formulas, $[\alpha_i]\phi_i$, separately and combining the results conjunctively.

III. Problem Formulation

In this section, we use BTs to synthesize a sequence of actions for an autonomous-agent over the following components:

1) The term R represents an autonomous agent characterized by the position $P_r = (p_x, p_y)$ and velocity $V_r = (v_x, v_y, \omega_r)$ pairs, where p_x, p_y, v_x, v_y, ω_r are the position of R along the x-axis, the position of R along the y-axis, the linear velocity of R along x-axis, the linear velocity of R along y-axis, and the angular velocity of R, respectively. Here, the terms agents, robots, and vehicles are used interchangeably.

2) The set A is the agent action bank, which contains a set of actions A_k, $k = 1, \cdots, L$, where $L \in \mathbb{N}$ is the total number of actions. We also define a safety constraint associated with each action, $\phi_{A_{sk}}$, $k = 1, \cdots, L$. Here, the robot is assumed to perform a single action at a time.
3) The set T which includes a set of complex tasks T_j, $j = 1, \cdots, N$, where $N \in \mathbb{N}$ is the number of tasks. Each task T_j is associated with a safety property $\phi_{T_{sj}}$, $j = 1 \cdots, N$. Furthermore, the accomplishment of task T_j with safety constraint $\phi_{T_{sj}}$ can be captured by meeting a condition C_j. For example, if the task T_1 is to "reach a goal region" with safety constraint $\phi_{T_{sj}}$ "avoid all obstacles", then C_1 is "being at the goal region." To reach the "goal" of a task T_j, depending on the agent's capability, a series of actions from the action bank A should be completed, where the last action should meet C_j.
4) We define the set $\hat{C}$ which includes a set of preconditions $\hat{c}_{kp}$, $k = 1, \cdots, L$, and $p = 1, \cdots, P_k$, where P_k is the number of preconditions for action A_k, and $\hat{c}_{kp}$ specifies pth preconditions for completing action A_k by robot R.
5) We define an operation $R \models con$ which cheeks if the agent R at its current state satisfies the condition con, where the condition con can be a condition for a task, i.e., $C_j, j = 1, \cdots, N$, or a precondition for an action $\hat{c}_{kp}$, $k = 1, \cdots, L$, and $p = 1, \cdots, P_k$. We also redefined the satisfiability operation for safety constraint where $\phi_{A_{sk}} \models \phi_{s(k+*)}$, $k, * = 1, \cdots, L$, implies that the state of robot R after the execution of action A_k, satisfies the safety requirement of action $A_{(k+*)}$. (As a trivial assumption, each individual action A_k should not violate the task safety requirement $\phi_{T_{sj}}$).

Now the safe coordination of an agent can be described as:

Problem 1: Consider a $Mission$, which consists of several tasks T_j, $j = 1, \cdots, N$, associated with a safety constraint $\phi_{T_{sj}}$, $j = 1, \cdots, N$, to be completed by a robot R, that is capable of accomplishing actions A_k, $k = 1, \cdots, L$. Then, synthesize a BT for the robot R to meet the goal, C_j, of a set of tasks T_j while simultaneously satisfying the safety constraint $\phi_{T_{sj}}$ of each task.

IV. Automatic Safe Behaviour Tree Synthesis

To address Problem 1, we propose to generate a sequence of actions using Algorithm 1 to meet the goal of the mission and satisfy the safety requirements. Given a mission in the form of a set of tasks T_j, $j = 1, \cdots, N$, we generate the BT for each task to satisfy the mission goal and meet the safety requirements by selecting verified safe actions, A_k, $k = 1, \cdots, L$.

A. Modular Synthesis of a BT

The process of generating a safe BT for a task T_j is given in Algorithm 1. Algorithm 1 starts from the "goal" input, which is described as the accomplishment of a "task" (Line 4). The algorithm then iteratively executes and updates the BTs until a sequence of actions is obtained for the task T_j, which together realize the goal of the task, C_j (Lines 6 to 15). As long as the condition $\neg(R \models C_j)$ is true, the BT is tested to determine whether it is executable or not (Lines 7-9). If not, Line 10 identifies the *cause*, c_f. The identified *cause* will become a condition in a sub-tree to resolve the problem by finding alternative actions (Line 11) to update the BT, as it will be described in Algorithm 2. However, after updating the BT, due to the addition of a new sub-tree, $\mathcal{T}_{subtree}$, a conflict could arise. To resolve the conflict, the function $Conflict(\mathcal{T}_i)$ increases the priority of $\mathcal{T}_{subtree}$ by moving the subtree toward the left, e.g., opening a door should be done when the robot arm is free either by putting down the object or at a time that the arm is free (Lines 12-14). Once, the goal of a task is achieved, i.e., the condition $(R \models C_j)$ is met, the BT for the next task has to be initialized (Lines 4-5) by incrementing the task index "i" (Line 16).

Algorithm 1: Main BT Synthesis and Execution

```
1  function Main BT Synthesis and Execution (T) ;
   Input  : T : set of tasks
   Output: T = Operational BT
2  i =1 // Set task index to 1
3  while T[i] ≠ NULL do
4      C_j = T[i] //goal of task i
5      T ← C_j, i = 1 // Initialize the tree
6      while ¬(R ⊨ C_j) do
7          do
8              r ← Execute(T)
9          while r = Executable;
10         c_f ← GetConditionToExpand(T)
               //Identify the cause for not being executable
11         T, T_subtree ← ExpandBTSafely(T, c_f, φ_Ask)
               //Resolve the cause by Algorithm 2
12         while Conflict(T) do
13             T ← IncreasePriority(T_subtree)
14         end
15     end
16     i = i+1 // update the task index
17 end
```

As mentioned in the explanation of Algorithm 1, when a task is not executable by an agent due to a condition (*cause*), c_f, Algorithm 2 will synthesize a sub-tree by identifying proper actions from the agent's action bank. In Line 2 of Algorithm 2, a set of actions LA are identified that could satisfy the condition c_f. In a loop (Lines 4-16), a safety constraint test is performed for each suitable action $A_k \in LA$, i.e., we check if the selected action does not violate the preconditions of the successor action (Line 5). Then, the action that meets the safety criteria along with its preconditions, $\hat{c}_{kp}$, are composed by a sequence node to from $\mathcal{T}_{seq}$ (Lines 8 to 12). Further, $\mathcal{T}_{seq}$ is composed with $\mathcal{T}_{sel}$, defined as c_f (Line 3), by a selector node, to enforce the

execution of $\mathcal{T}_{seq}$ only in situations where c_f is not satisfied (Line 14).

Algorithm 2: Expanding the Behavior Tree

1 function ExpandBTSafely $(\mathcal{T}, c_f, \phi_{s\_sucessor})$;
  Input : $\mathcal{T}$ = the BT to be expanded, c_f = condition (cause) for not being executable, $\phi_{A_{s\_sucessor}}$ = successor action safety constraint
  Output: $\mathcal{T}$ = Safely Expanded BT
2 $LA \leftarrow GetActionwithPrecondtion(c_f)$
  //Identify actions that satisfy c_f
3 $T_{sel} \leftarrow c_f$
4 **for** $A_k \in LA$ **do**
5 **if** $\phi_{A_{sk}} \models \phi_{A_{s\_sucessor}}$ **then**
6 // Action is selected only if successor safety
7 is not violated by action A_k
8 $\mathcal{T}_{seq} \leftarrow \emptyset$
9 $\hat{c}_k = GetPrecondtionforAction(A_k)$
10 **for** $\hat{c}_{kp}$ *in* $\hat{c}_k$ **do**
11 $\mathcal{T}_{seq_i} \rightarrow Sequence(\mathcal{T}_{seq}, \hat{c}_{kp})$
  //sequence BT with the condition of action
12 **end**
13 $\mathcal{T}_{seq} \leftarrow Sequence(\mathcal{T}_{seq}, A_k)$
  // Generate a sequence subtree containing action A_k and its preconditions
14 $T_{sel} \leftarrow Selector(T_{sel}, T_{seq})$
15 $break$ // Suitable action is found
16 **end**
17 **end**
18 $\mathcal{T} \leftarrow Substitute(\mathcal{T}, c_f, \mathcal{T}_{sel})$ // add the subtree to T
19 return $\mathcal{T}$, $\mathcal{T}_{sel}$

B. Modelling a BT Using DL Operations

In this section, we map each BT node to appropriate DL operations, which enables us to take advantage of compositional verification of DL and apply it to the modular structure of BTs. The *Sequence* node is equivalent to a sequence operation in DL, where the HP $[\alpha; \beta; \cdots]$ represents the execution of actions $\alpha, \beta, \cdots$ in sequence. The *Selector* node does not have an equivalent DL operation, but it can be modeled by the nondeterministic DL operator $\cup$. The difference between a *Selector* node and a nondeterministic operation is that in a *Selector* node, actions are selected in a sequence with the highest priority action first, while for the nondeterministic DL operator, actions are selected randomly. Even if from an operation point of view, they are not the same, from a safety point of view, the nondeterministic DL operator over-approximates the *Selector* node. Hence, a *Selector* node can be modeled by a nondeterministic DL operation $[\alpha \cup \beta]$ where α or β is executed randomly. The action node directly represents a verified HP, $[\alpha]$. For our case, all HP actions have to be safe. The condition node can be modeled by a conditional DL operation $[?\chi]$, where execution can only proceed if χ is true. Assuming that there is no dependency among the actions, the parallel node can be modeled by a sequential operation $[\alpha; \beta]$ where β is executed after α. The relation between the BT nodes and their equivalence DL operation is given in Table I.

C. Safety Verification of a BT Modelled Via DL

Next, we show that the process for generating BTs in Algorithm 1 respects safety requirements.

Theorem 1: Consider an autonomous agent R with a set of actions A_* which respect the safety constraint $\phi_{A_{s*}}$. Using Algorithm 1 to synthesize a BT for a task T_j, by composing these safe actions, respects the task's safety requirement, $\phi_{T_{sj}}$.

Proof: To generate a BT Algorithm 1 creates a sequence of actions, which can be modelled by DL operations as $\phi_{T_{sj}} \rightarrow [?(\phi_{A_{sk}}); A_k; ?(\phi_{A_{sk}}); ?(\phi_{s(k+*)}); A_{k+*} \cdots]\phi_{T_{sj}}$ where $k, * \in 1 \cdots N$, and $(k + *) \leq N$. As shown in [16], applying the rules for $[;]$, $[?]$ and $\rightarrow$, the sequence of actions can be modelled as $\phi_{A_{sk}} \rightarrow [A_k]\ \phi_{A_{sk}}$ where $?(\phi_{A_{sk}})$ and $?(\phi_{s(k+*)}$ are the same or $?(\phi_{A_{sk}})$ does not violate $?(\phi_{s(k+*)})$. Then, by applying the DL compositional property, as long as the execution of each action or operation starts from a safe region, $?(\phi_{A_{sk}})$, and it does not violate the safety property at completion, $\phi_{A_{sk}} \rightarrow [A_k]\phi_{A_{sk}}$, the generated BT is guaranteed to be safe with respect to the safety requirement, $\phi_{T_{sj}}$. ■

Theorem 2: A safe BT is synthesizable for meeting the goals of any task defined over the action bank A if for any task/action (successor), (a) there is at least one action (predecessor) for accomplishing each successor's precondition, and (b) the execution of the predecessor action does not violate the safety requirement of the successor.

Proof: In [17], it is shown that if there is at least one action (predecessor) for accomplishing each successor's precondition, Algorithm 1 terminates in finite time and generates a dead-lock free, live-lock free, and finite time BT that satisfies the goal, C_j, of the task T_j.

Further, based on Theorem 1, if the execution of the predecessor action does not violate the safety requirement of the successor, i.e., $?(\phi_{A_{sk}})$ does not violate $?(\phi_{s(k+*)}$, the synthesized BT respects the task's safety requirement, $\phi_{T_{sj}}$. ■

V. Safety Verification of Required Actions

In this section, we explain the synthesis of safe actions needed for a search and deliver mission that will be described in Section VI.

A. MoveTo (A_1)

The navigation function for the robot is handled by the *MoveTo* action which takes the destination as the input parameter. We adopt the formulation from [18] and implement the navigation function based on dynamic window approach (DWA). The DWA algorithm generates a safe trajectory at every time instant that is uniquely identified by the linear velocity v_r and angular velocity ω_r based on the current location of the autonomous vehicle. Without going into the details, DWA generates safe trajectories in two steps: (i) first,

TABLE I

BTs NODE AND THE CORRESPONDING DL REPRESENTATION TYPES

No	Node Type	DL equivalence	Description
1	Sequence	$\alpha;\beta$	execute HP α and β in sequence
2	Selector	$\alpha \cup \beta$	non-deterministically execute HP α or β
3	Action	α	a HP representing an action
4	Condition	$(?\chi;\alpha) \cup \neg\chi$	execute action α if condition χ is true, otherwise it has no effect

by considering the dynamics of the robot, it identifies a set of (v_r, ω_r) pairs that are safe and feasible to be executed in a short period of time, and (ii) it optimizes an objective function, which takes into account the progress towards the goal, forward speed of the agent, and the next obstacle along with the robot's trajectory.

In [19], the passive safety property (the ability of an agent to safely stop before colliding with an obstacle) of DWA has been proven. Hence, no additional safety verification is needed. However, to generate a safe trajectory, the agent must start operating in a safe region. Formulating this requirement by DL operations, we have $\phi_{A_{s1}} \rightarrow [MoveTo]\phi_{A_{s1}}$, which means that if the agent starts from a region that satisfies $\phi_{A_{s1}}$, then the action $MoveTo$ always preserves the safety constraint, $\phi_{A_{s1}}$.

B. Detect (A_2)

The action *Detect* does not affect the state of the autonomous vehicle or the environment that the agent is operating, hence its safety is not required to be verified. However, for the mission to be successful, the detection probability has to be high.

C. PickUp (A_3)

The *PickUp* action performs reaching, grasping, and lifting operations in sequence or in order to pick an object. Since the pickup operation does not depend on the kinematics of the agent or the environment, verification is not required. As a precondition for this action, "the agent should not carry another object" and "the object of interest has to be identified first."

D. Deliver (A_4)

Similar to the $Pickup$ action, the $Deliver$ action performs reaching and placing operations in sequence to place an object at a predefined location. Again, the safety property for this controller does not depend on the agent kinematics of the robot, as such, there is no need for verification. Nonetheless, for the $Deliver$ action to succeed, the agent has to initially "carry the object of interest".

E. Search (A_5)

The $Search$ action is defined as a parallel composition of the $Detect$ and $Moveto$ actions. As the name implies, the purpose of the search action is to locate an object by moving around and performing detection while avoiding obstacles ($\phi_{A_{s5}}$ = avoid obstacle). Using DL terminologies, this can be modeled as a sequential operation $Moveto$ followed by $Detect$, executed repeatedly until the object of interest is detected, i.e., $Model$ 1: $[?(\neg obj); t := 0; t' = 1; MoveTo; Detect \& (t <= 1)]^*$, where the $Moveto$ and $Detect$ actions are executed in sequence for at most 1 sec as expressed by the evolution domain $\&(t <= 1)$ until the object of interest is detected. When the object of interest is detected the DL operation $?(\neg obj)$ becomes false and the execution stops. Since the search terminates in a finite time, $Model$ 1 can be represented as $\phi_{A_{s5}} \rightarrow [MoveTo_1, Detected_1; MoveTo_2; Detect_2; \cdots]$ $\phi_{A_{s5}}$ which can be reduced to $\phi_{A_{s5}} \rightarrow [Moveto_i]\phi_{A_{s5}} \wedge \phi_{A_{s5}} \rightarrow [Detect_i]\phi_{A_{s5}}$. Therefore, since both actions are safe, the search action is also safe.

Table II provides a summery of the basic actions, the precondition for each action and the effect of the actions.

TABLE II

ACTION BANK ALONG WITH THEIR PRECONDITIONS AND THE EFFECTS FOR THE CASE STUDY VI

Safe Action Bank			
Actions	Description	Precondition	Effect
A_1	MoveTo $(Np, path)$	$intially\ path\ is\ collsion free$	$R\ at\ Np$
A_2	Detect(m)	-	$m\ is\ detected$
A_3	Deliver(o, p)	$R\ at\ Np$ $m\ is\ detected$ $o\ is\ at\ R\ arm$	$o\ at\ p$
A_4	Pick(o)	$arm\ is\ free$ $o\ is\ detected$	$o\ is\ at\ R\ arm$
A_5	Search(o)	$intially\ path\ is\ collsion free$	$o\ is\ detected$

VI. CASE STUDY

Consider a search and delivery mission in which the objective is to deliver an object o to a specific place marked by m near position p. A robot R has to locate and pick the object o and search for the marking m in the close vicinity of p, N_p, before delivering the object o. Then, the problem is that given the verified actions in Table II, we should generate a safe BT using Algorithms 1 and 2 to achieve the task.

Algorithm 1 starts from the goal, "o at p", i.e., the object o should be at position p, as shown in Figure 2a. Since, initially the goal is not satisfied and the generated BT (Line 8 of Algorithm 1) is not executable, the function $GetCondtionsToExpand$ is called to identify the preconditions (Line 10 of Algorithm 1). From Table II, the $Deliver$ action can meet the precondition, where the $ExpandBtSafely$ function (Line 11 of Algorithm 1) uses this action to update the BT by composing the conditions of $Deliver$ action via a sequence node and the goal by a selector node (Lines 8-14 of Algorithm 2) as shown in Figure 2b. The expansion process is repeated and Figure 2c shows the final BT after expanding the conditions that are not satisfied.

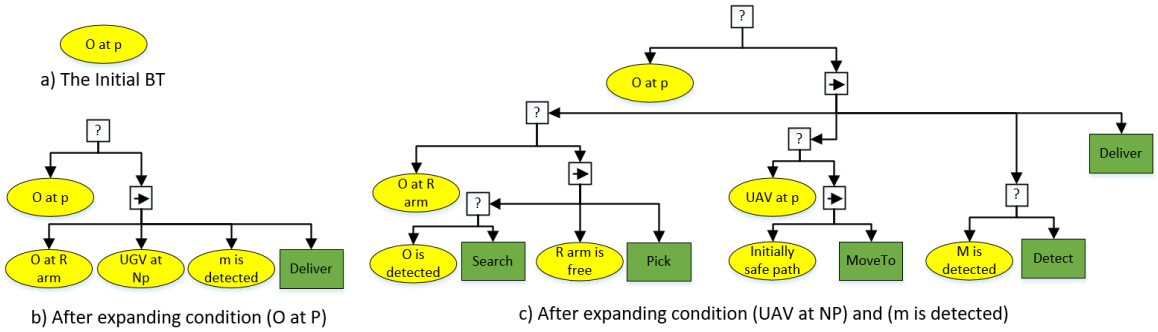

Fig. 2. Synthesizing a BT for the robot R to search and deliver an object to a particular position: (a) Algorithm 1 starts with the goal condition "o at p", (b) Algorithm 2 identify the action "Deliver" which can meet the goal condition, and hence, the action " deliver" and its preconditions (being at Np, detecting m, and O at R arm) are added as a subtree with a selector node, (c) Algorithm 2 expands the false preconditions (being at Np, detecting o, and detecting m) to find the actions that can meet these preconditions.

Now, for a given task consisting of these actions, once the BT is generated, we can map the corresponding nodes to the DL operation and perform DL composition to guarantee the safety of the generated tree. However, based on Theorem 1, since the basic actions are safe, the synthesized BT is safe hence, no further verification is needed. Also, according to Theorem 2, the generated BT can meet the goals of the task. Therefore, the BT can both simultaneously satisfy the goal and safety requirements of a task.

VII. CONCLUSION

This paper developed a new automatic safe BT synthesis technique to coordinate an autonomous agent, which has different capabilities in terms of executing different tasks. In the proposed framework, the synthesized BT acts as a high-level planner that sequences verified actions to meet the mission goal. Further, to verify the safety of the system (synthesized BT) from its components, the compositional verification property of DL was employed. This allowed us to decouple the synthesis of a BT from verification of its components. Future work includes implementing the developed framework and extend the method to multi-agent systems.

ACKNOWLEDGMENT

The authors would like to acknowledge the support from the National Science Foundation under the award number 1832110 and Air Force Research Laboratory and OSD under agreement number FA8750-15-2-0116.

REFERENCES

[1] B. Xin, G. Q. Gao, Y. L. Ding, Y. G. Zhu, and H. Fang, "Distributed multi-robot motion planning for cooperative multi-area coverage," in *2017 13th IEEE International Conference on Control Automation (ICCA)*, July 2017, pp. 361–366.

[2] L. Shamgah, T. G. Tadewos, A. Karimoddini, and A. Homaifar, "Path planning and control of autonomous vehicles in dynamic reach-avoid scenarios," in *2018 IEEE Conference on Control Technology and Applications (CCTA)*, Aug 2018, pp. 88–93.

[3] A. Ulusoy, S. L. Smith, X. C. Ding, C. Belta, and D. Rus, "Optimality and robustness in multi-robot path planning with temporal logic constraints," *The International Journal of Robotics Research*, vol. 32, no. 8, pp. 889–911, 2013.

[4] L. Shamgah, T. G. Tadewos, A. Karimoddini, and A. Homaifar, "A symbolic approach for multi-target dynamic reach-avoid problem," in *2018 IEEE 14th International Conference on Control and Automation (ICCA)*, June 2018, pp. 1022–1027.

[5] P. Ramadge and W. Wonham, "The control of discrete event systems," vol. 77, no. 1, pp. 81–98, 01 1989.

[6] M. Karimadini, A. Karimoddini, and H. Lin, "Modular cooperative tasking for multi-agent systems," in *2018 IEEE 14th International Conference on Control and Automation (ICCA)*, June 2018, pp. 618–623.

[7] M. Karimadini, H. Lin, and A. Karimoddini, "Cooperative tasking for deterministic specification automata," *Asian Journal of Control*, vol. 18, no. 6, pp. 2078–2087, 2016.

[8] M. Darrah, W. Niland, and B. Stolarik, "Multiple uav dynamic task allocation using mixed integer linear programming in a sead mission," in *Infotech@ Aerospace*, 2005, p. 7164.

[9] A. Marzinotto, M. Colledanchise, C. Smith, and P. gren, "Towards a unified behavior trees framework for robot control," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, May 2014, pp. 5420–5427.

[10] T. G. Tadewos, L. Shamgah, , and A. Karimoddini, "On-the-fly decentralized tasking of autonomous vehicles," in *Proc. of 58th IEEE Conference on Decision and Control (CDC)*, 2019.

[11] M. Colledanchise, R. M. Murray, and P. Ögren, "Synthesis of correct-by-construction behavior trees," in *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*. IEEE, 2017, pp. 6039–6046.

[12] P. A. Lindsay, K. Winter, and N. Yatapanage, "Safety assessment using behavior trees and model checking," in *2010 8th IEEE International Conference on Software Engineering and Formal Methods*. IEEE, 2010, pp. 181–190.

[13] R. J. Colvin and I. J. Hayes, "A semantics for behavior trees using csp with specification commands," *Science of Computer Programming*, vol. 76, no. 10, pp. 891–914, 2011.

[14] N. P. Yatapanage, *Slicing Behavior Trees for verification of large systems*. Griffith University, 2012.

[15] R. R. da Silva, B. Wu, and H. Lin, "Formal design of robot integrated task and motion planning," in *2016 IEEE 55th Conference on Decision and Control (CDC)*. IEEE, 2016, pp. 6589–6594.

[16] A. Platzer, *Logical analysis of hybrid systems: proving theorems for complex dynamics*. Springer Science & Business Media, 2010.

[17] M. Colledanchise, D. Almeida, and P. Ogren, "Towards blended reactive planning and acting using behavior trees," *arXiv preprint arXiv:1611.00230*, 11 2016.

[18] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics & Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.

[19] S. Mitsch, K. Ghorbal, D. Vogelbacher, and A. Platzer, "Formal verification of obstacle avoidance and navigation of ground robots," *The International Journal of Robotics Research*, vol. 36, no. 12, pp. 1312–1340, 2017.