Expert Systems With Applications 201 (2022) 117022

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

Specification-guided behavior tree synthesis and execution for coordination of autonomous systems☆

Tadewos G. Tadewos, Abdullah Al Redwan Newaz, Ali Karimoddini *

Department of Electrical and Computer Engineering, North Carolina A&T State University, NC, USA

ARTICLE INFO

Keywords:
Behavior tree
Linear temporal logic
Formal methods
Autonomous control
Correct-by-design control
Autonomous Vehicles

ABSTRACT

This paper develops an automatic online Behavior Tree (BT) synthesis and execution technique to guide an autonomous agent to accomplish a series of missions expressed in Fragmented-Linear Temporal Logic (F-LTL). For this purpose, a novel top-down, divide-and-conquer method is developed to decompose the original F-LTL formulas into simpler sub-formulas, followed by synthesizing the corresponding sub-BTs. Then, the safe and reachable regions are calculated to identify the winning set for the sub-BTs and the associated winning paths. If the computed winning set is non-empty, the sub-BTs are composed to form a coordinator whose execution guarantees the satisfaction of the original F-LTL formulas. The correctness of the proposed method is proved. Unlike most existing methods which manually design BTs and suffer from high computation cost, the proposed method can automatically synthesize the BTs on-the-fly for F-LTL formulas with polynomial complexity in the size of the formula and the environment. The developed method is applied to several scenarios with different missions and sizes of the environment using a physics-based simulator. The simulation results demonstrate the capability of the proposed method to handle missions described by F-LTL formulas and the scalability of the approach in terms of the size of the environment.

1. Introduction

A fundamental challenge for coordination of autonomous systems is to design a mission-driven controller to satisfy desired specifications. Conventional engineering practice is to employ a bottom-up approach by designing a controller followed by testing and evaluating it through an extensive set of simulations and experiments to identify possible problems in the system, and then, redesigning the system to fix the problems and satisfy the design requirements (Lee & Yannakakis, 1996). However, this trial-and-error approach with ad hoc requirement refinement is costly and time consuming, and in the end, there is no guarantee to achieve the desired performance.

On the contrary, top-down approaches aim to develop correct-by-design controllers which provide a guaranteed performance against desired specifications (Wu et al., 2015). A common specification-guided top-down approach is to design coordinators to achieve high-level specifications given in the form of a regular language (Feng & Wonham, 2008; Wang, Moor, & Li, 2020; Wang, Wang, & Li, 2020) or a Temporal Logic formula (Camacho et al., 2018; Kloetzer & Mahulea, 2015; Raman et al., 2015). The resulting coordinator is often an automaton which produces sequences of high-level actions. The generated actions are then executed by translating them to continuous motion primitives. These methods, however, lack scalability due to the high computation cost for large environments with a high number of partitions. More importantly, these methods suffer from the lack of flexibility in the sense that the specifications and the environment should be fully known before designing the coordinator, and any change in the specification may require redesigning the coordinator from the scratch.

In practice, however, a system may need to be involved in a set of tasks, each described by high-level specifications that may be introduced to the system as the mission is evolved, not necessarily before the mission starts. In this case, the aforementioned offline methods for designing the coordinators are not applicable to such a scenario. Reactive temporal planning methods include receding-horizon GR(1) approach for controller synthesis (Maoz & Shevrin, 2020; Shamgah et al., 2018; Wongpiromsarn et al., 2012), revising unsatisfied temporal logic expressions (Fainekos, 2011; Guo et al., 2013), and counterexample-guided supervisor synthesis (Lin & Hsiung, 2011; Wu & Lin, 2016) may handle limited changes in the specification or the environment but often at a high computation cost of controller synthesis.

Motivated by these challenges, we are interested in addressing the problem of designing a computationally-effective and scalable coordinator for an autonomous system to achieve a sequence of missions in

☆ This document is the results of the research project funded by the Air Force Research Laboratory and OSD under agreement number FA8750-15-2-0116.
* Corresponding author.
E-mail addresses: tgtadewos@aggies.ncat.edu (T.G. Tadewos), aredwannewaz@ncat.edu (A.A. Redwan Newaz), akarimod@ncat.edu (A. Karimoddini).

https://doi.org/10.1016/j.eswa.2022.117022
Received 5 July 2021; Received in revised form 13 November 2021; Accepted 27 March 2022
Available online 9 April 2022

the form of high-level specifications which are introduced to the system on-the-fly, while respecting the safety requirements. As an example, consider an Intelligence, Surveillance, Target Acquisition, and Reconnaissance (ISTAR) mission in which a robot should search different assigned areas to find targets and collect the required information, while avoiding restricted zones and obstacles based on the terrain information. Certainly, offline planning cannot be a solution for the coordination of the robot for such a complex mission with random arrival of tasks and their location. To address this problem, we propose to employ Behavior Trees (BTs), which provide a hierarchical, modular, and reactive task execution models (Agis et al., 2020; Colledanchise & Ögren, 2017; Iovino et al., 2020; Kim et al., 2012). With BTs, it is more convenient to manage, modify, and add tasks or subtasks due to their modular and scalable structure. BTs have been applied for many applications, including but not limited to robotic manipulation (French et al., 2019), controlling an aerial vehicle (Scheper et al., 2016), coordination of ground robots (Coronado et al., 2019), etc.

Despite the great capabilities of BTs, there are only a few works in the literature about formal, systematic, and specification-driven synthesis of BTs. Most existing results manually design BTs (Guerin et al., 2015; Ruifeng et al., 2019). In Nicolau et al. (2017), a game theory approach, in Dey and Child (2013) a Q-learning method, and in Colledanchise et al. (2019) a reinforcement learning technique is used for automatic synthesis of BTs. However, these methods lack proof of guaranteed performance and are often applicable to simple specifications in the form of a single proposition such as reaching a particular state or meeting a certain condition. In our earlier work (Tadewos et al., 2019a, 2019b), we have shown a safe BT can be automatically synthesized using Dynamic Differential logic (DL) by concatenating a safe set of actions under the assumption that the completion of each action meets the safety requirement of its successor action. This condition is not practically feasible to meet and verify in many cases such as the situations that require synthesizing a BT for an action with a preposition that requires multiple sequences of actions. In Tumova et al. (2014), a maximally satisfying LTL action planning framework is proposed that utilizes BTs as a middle layer which interfaces the high-level discrete planner with the low-level continuous controller. Given a robot and a specification in the form of State/Event Linear Temporal Logic (SE-LTL), this framework generates a high-level planner as a weighted product automaton. However, using BTs as a middle layer in conjunction with the generated product automaton increases the computational complexity and challenges the scalability of this method.

To address these challenges, in this paper, we develop an automatic BT synthesis and execution method to coordinate a robot to meet a series of missions. We leverage the rich expressivity of Fragmented Linear Temporal Logic (F-LTL) to capture mission requirements including response/reactivity, safety, reachability, and recurrence (infinitely often). The proposed approach is illustrated in Fig. 1. With a "divide-and-conquer" strategy, we decompose the original F-LTL formulas into simpler sub-formulas, for which we synthesize separate BTs. For each of these BTs, we compute the safe and reachable sets, the intersection of which form the winning set. In parallel, the paths to reach the winning set will be calculated. If the winning set is not empty, these BTs can be composed to realize the original F-LTL specification, which can serve as a high-level planner to generate a sequence of actions as well as safe paths to reach the winning set. In summary, the main contributions of this paper are:

- Developing a top-down, correct-by-design, divide-and-conquer approach for automatic online behavior tree synthesis and execution to coordinate a robot to meet a series of desired missions described by F-LTL specifications.
- Computing a winning set for the synthesized BTs from which, we can calculate a set of paths whose execution satisfies the missions given in the form of F-LTL specifications.

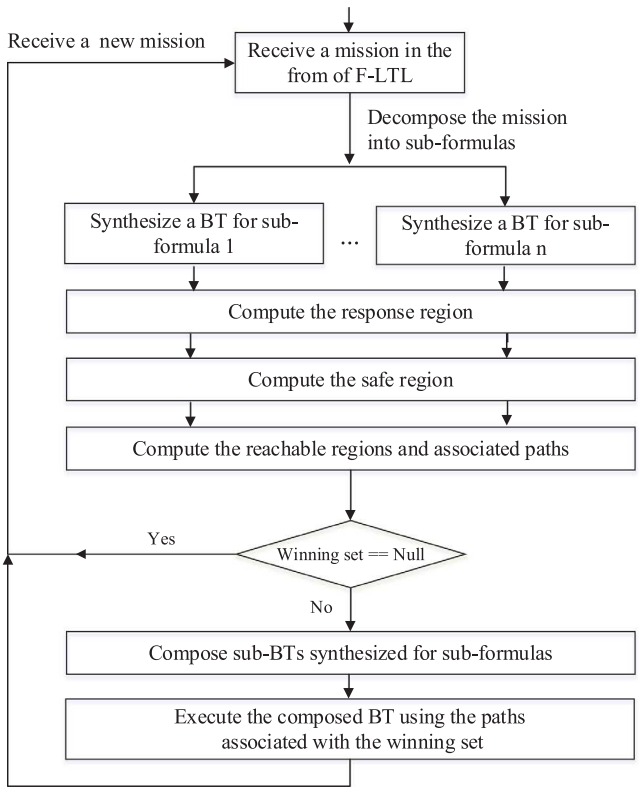


Fig. 1. The overview of the proposed method for BT synthesis and execution to satisfy missions given in F-LTL.

- Proving the correctness of the proposed method along with the analyzing the computational complexity of the developed algorithms.
- Verifying the developed method via a physics-based robotic simulator for handling the assigned missions.

Compared to the manual approach for designing BTs in Guerin et al. (2015), our proposed method synthesizes BTs automatically. Compared to AI-based techniques in Colledanchise et al. (2019), Dey and Child (2013), Nicolau et al. (2017), our approach guarantees that the execution of the synthesized BTs satisfies the desired specifications. Further, unlike many existing approaches where synthesizing an action policy from LTL specifications suffers from double-exponential or exponential time complexity (Kloetzer & Mahulea, 2015; Pnueli & Rosner, 1989), focusing on F-LTL formulas, the proposed method features polynomial time complexity. The proposed method in Colledanchise et al. (2017), synthesizes BTs by computing a value function, requirement function, and constraint function for each formula to generate the BT from F-LTL specifications. Specifically, the requirement function in Colledanchise et al. (2017), is given as a transition rule, which is then used to identify the actions that have to be performed by a BT as middle layer. In comparison, our proposed method utilizes a uniform BT synthesis algorithm to generate the sequence of actions to meet a given specification, allowing for accommodation of more complex missions. Further, our hierarchical approach allows the treatment of the computation of the winning set and the synthesis of the BT separately. The manageable computation cost and the adopted hierarchical and "divide-and-conquer" approach explain the scalability of the proposed method.

The rest of the paper is organized as follows. The background and necessary preliminaries on behavior trees are provided in Section 2. In Section 3, the problem to synthesize a BT from the F-LTL specification is formulated. Section 4 describes our proposed approach for synthesizing

the BTs and identifying the winning set in detail. Section 5 provides the proof of correctness and complexity of the developed algorithms. Section 6 applies the proposed method to a search, delivery, and patrolling case study to illustrate the implementation of the developed algorithms. Also, the computation time for different steps of the proposed method and for different sizes of the environment are calculated. Finally, Section 7 concludes the paper.

2. Preliminaries

This section provides the preliminaries and notations that are needed for developing the proposed framework including BT formalism and F-LTL syntax and semantics.

2.1. Behavior Tree (BT)

Behavior Tree (BT) is a graphical modeling language that executes a task by controlling the decision making process of a robot to achieve the desired goal. Structurally, a BT is a directed graph in which each node represents either a leaf node or a composite node with the *root* node at the top of the tree. A root node has no parent, a composite node has both a parent and one or more child(ren), while a leaf node has only a parent node.

Leaf nodes are terminal nodes with the ability to observe the environment (*condition* nodes) or to act on the environment (*action* nodes). A *condition* node returns either *success* or *failure*, by evaluating the actual observations. An action node acts on the environment and returns *success* if the action is completed, *failure* if the action is failed, and *running* if the action is in progress.

Composite node organizes multiple nodes under a single parent to achieve a unique behavior. For example, a *sequence* node is used to execute its children nodes in sequence. A *selector* node is used to execute its children nodes in order only if the previous nodes fail. A *sequence* node returns *success* only if all the composed nodes return *success*, otherwise it returns *failure*. On the other hand, a *selector* node returns *success* as soon as one of its child nodes returns *success*, otherwise it returns *failure* if all the composed nodes return *failure*. Both *sequence* and *selector* nodes return *running* if one of their children nodes is in progress. Figs. 2.a, 2.b, and 2.c show a pair of condition and action composed by a *selector* node, a set of actions composed by a *sequence* node, and a set of actions composed by a *selector* node, respectively.

The execution of a BT starts from the *root* node. Then, the *root* node sends a tick (enabling signal) to its children such that nodes are traversed from top to bottom and left to right. The tick flows from parents to their child(ren) until a terminal node is reached. Then, the terminal node is executed. Thus, with this process, the actions are executed starting from the bottom left of the BT.

By the proper combination of leaf nodes (actions and condition nodes), and composite nodes (*sequence* or *selector* nodes), a complex BT structure can be modularly synthesized to effectively meet the goal of a mission.

2.2. Finite state transition system

A transition system can describe discrete changes over the states of a system and can be formally defined as:

Definition 1. A transition system is a tuple $TS = (S, \Sigma, \delta, S_o, AP, L)$ where S is a finite set of states, Σ is a finite set of actions, δ is a transition relation $S\ x\ \Sigma \rightarrow 2^S$, S_o is an initial state $\in S$, AP is a set of atomic propositions, and L is a mapping function $L : S \rightarrow 2^{AP}$.

A run of TS is defined as a finite/infinite sequence of states $\sigma = s_0, s_1, \ldots$ where $s_0 = S_0$, $s_k \in S$ and $(s_k, s_{k+1}) \in \delta$ for all $k \geq 0$. For the run σ, a word $\omega = L(s_0), L(s_1), \ldots$ is generated, where $L(s_k)$ is a mapping function for the set of atomic propositions satisfied at state s_k. From a state $s \in S$ the number of successors could be one (deterministic, i.e. $|\delta(s, \Sigma)| = 1$) or more than one (non-deterministic, i.e., $|\delta(s, \Sigma)| > 1$). We assume the system under consideration is non-blocking, i.e., $|\delta(s, \Sigma)| \geq 1$.

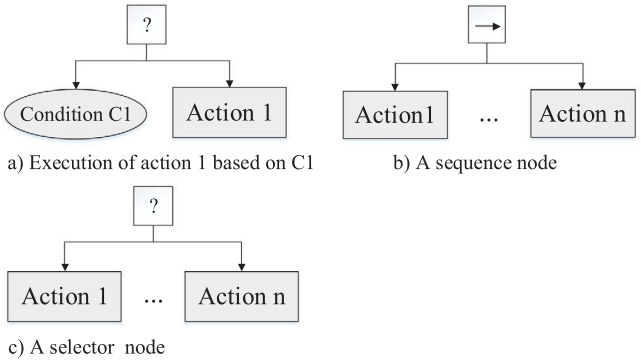


Fig. 2. Building blocks of Behavior Trees.

Given a system expressed as a transition system, we define the operator $Cpre(.)$ to facilitate the computation of the winning region as follows:

Definition 2. Controlled predecessor $CPre(f) := \{b \in S \mid \exists a\ s.t.\ \delta(b, a) = \{f\}\}$, where $CPre(f)$ returns the set b that always reach state f for action a.

2.3. Fragmented linear temporal logic (F-LTL)

To state the mission objective of an agent at a high level, we employ F-LTL (Wolff et al., 2013) that can describe properties like safe navigation, immediate response, coverage, and surveillance. While LTL is a more expressive language, compared to the F-LTL (has a polynomial complexity), the complexity of synthesizing a control policy is doubly-exponential in the length of the system (Pnueli & Rosner, 1989). The syntax and semantics of F-LTL are as follow:

$$\phi = \phi_A \wedge \phi_R \wedge \phi_F \wedge \phi_{AF} \tag{1}$$

where

$$\phi_A = \bigwedge_{i \in I_A} \square p_i$$

$$\phi_R = \bigwedge_{i \in I_R} \square (q_i \implies \bigcirc p_i)$$

$$\phi_F = \bigwedge_{i \in I_F} \lozenge p_i$$

$$\phi_{AF} = \bigwedge_{i \in I_{AF}} \square \lozenge p_i$$

In the above definition p_i is an atomic proposition under the control of the system, q_i is an atomic proposition controlled by the environment, I_A, I_R, I_F, and I_{AF} are finite set of indexes. Here, the sub-formulas $\square \lozenge p_i$ and $\lozenge p_i$ are assumed to describe reaching a destination which can be achieved by proper motion planning over a partitioned environment. Given a run $\sigma = (s_0, s_1, \ldots)$, the semantics of F-LTL is given as follows:

$\sigma \vDash \square p_1$, if and only if $\forall i\ s_i \vDash p_1$ for

$\sigma \vDash \square (q_1 \implies \bigcirc p_1)$ if and only if $\forall i\ s_i \nvDash q_1$ or $s_{(i+1)} \vDash p_1$

$\sigma \vDash \lozenge p_1$ if and only if $\exists i, s_i \vDash p_1$

$\sigma \vDash \square \lozenge p_1$ if and only if $\forall i \geq 0, \exists j \geq i, s_i \vDash p_1$

The satisfaction of ϕ by a transition system TS is denoted by $TS \vDash \phi$, where a run $\sigma = (s_0, s_1, \ldots)$ meets all the sub-formulas of ϕ.

3. Problem formulation

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

1. The robot R which is capable of performing a set of actions. Here, the terms agent, robot, and vehicle are used interchangeably.
2. The set A is the action bank, which contains a set of actions A_k, $k = 1, \dots, L$, where $L \in \mathbb{N}$ is the total number of actions that the robot is capable of. We also define a set of preconditions for each action, $\hat{c}_{kp}$, $p = 1, \dots, P$ where $P \in \mathbb{N}$ is the total number of preconditions for action A_k. If all the preconditions of an action are satisfied, then the execution of the action has an effect, E_k. Here, the robot is assumed to perform a single action at a time.
3. The set ϕ includes missions ϕ_j, $j = 1, \dots, N$, where $N \in \mathbb{N}$ is the total number of missions. The mission ϕ_j is a conjunction of sub-formulas, ϕ_{jm}, $m = 1, \dots, N_j$, expressed as F-LTL. Each sub-formula ϕ_{jm} has to be satisfied by the execution of a set of actions and each action requires a set of conditions to be met. The mission ϕ_j is completed when all sub-formulas are satisfied. Here, we assume that the missions and the number of missions, N, are not initially known to the robot. Instead, the missions are streamed to the system and are served by the robot sequentially for each of which a BT is synthesized in an online way.
4. To capture the interactions between the robot R and the environment, we define the transition system $T_E = (S, A, \delta, s_0, \Pi, L)$ by partitioning the environment into ξ bounded regions where $S = \{s_0, s_1, \dots, s_\xi\}$ is a set of regions in $\mathbb{R}^2$; A is the event/action set; $\delta : S \times A \to 2^S$ is the transition relation; s_0 is the region that the robot R initially is located; The environment is assumed to be static and is represented by a set of atomic propositions $\Pi = \{\pi_0, \pi_1, \dots, \pi_\xi\}$, where π_i is true if and only if the robot R is at region s_i, and $L : S \to 2^{AP}$ is the label function.

Given R, A, ϕ, and T_E, our objective is to synthesize and execute BTs for the completion of missions expressed as F-LTL specifications. This has been formally formulated in the following problem:

Specification-guided BT Synthesis and Execution Problem: *Consider ϕ as a set of streamed missions ϕ_j, $j = 1, \dots, n$ which are described by F-LTL specifications defined over Π. Also, consider the robot R, which is capable of executing the actions A_k, $k = 1, \dots, L$, and its interactions within the environment is captured by T_E. For each of the missions ϕ_j, (1) Identify the set of regions W_j from which the robot can achieve ϕ_j. (2) Synthesize a BT for coordinating the robot R such that $BT_j \vDash \phi_j$.*

4. Automatic behavior tree synthesis and execution

To address the *Specification-guided BT Synthesis and Execution Problem*, we propose a framework that generates a high-level plan for the received missions. Fig. 1 shows our proposed framework which has three folds: the synthesis of BTs, the computation of the winning set, and the execution of BTs to satisfy the missions. Algorithm 1, receives the missions streamed to the robot and calls function `HighLevelBT-Synthesis` to synthesizing BT_j (Line 4). Consider the mission ϕ_j given as an F-LTL. With a divide-and-conquer strategy, we decompose ϕ_j into sub-formulas ϕ_{jm} in the form of $\Box C_{jm}$, $\Diamond C_{jm}$, $\Box\Diamond C_{jm}$, or $\Box(C'_{jm} \implies \bigcirc C_{jm})$, where C'_{jm} and C_{jm} are atomic propositions (showing accomplishment of an action or satisfying a condition). Then, for each mission, ϕ_j, a BT is synthesized by satisfying sub-formulas ϕ_{jm}. Once the BTs are synthesized, i.e., the high level planning is completed, the winning set, W_j, needs to be computed (Line 5). If the winning set W_j is empty, then BT_j is set to null, and the Mission Controller will be ready to receive a new mission (Lines 6–9). Otherwise, the synthesized BT will be executed (Line 10). The components of the proposed framework are discussed next.

Algorithm 1: `MissionController`

```
1  Main()
2      while True do
3          φ_j ← ReciveMission()
4          BT_j ← HighLevelBTSynthesis(φ_j)
5          W_j, Γ_ju = GetWinningRegion(φ_j, BT_j)
6          if W_j == ∅ then
7              BT_j ← ∅                          // unrealizable
8              continue
9          end
10         ExecuteBT(BT_j, Γ_ju)
11     end
```

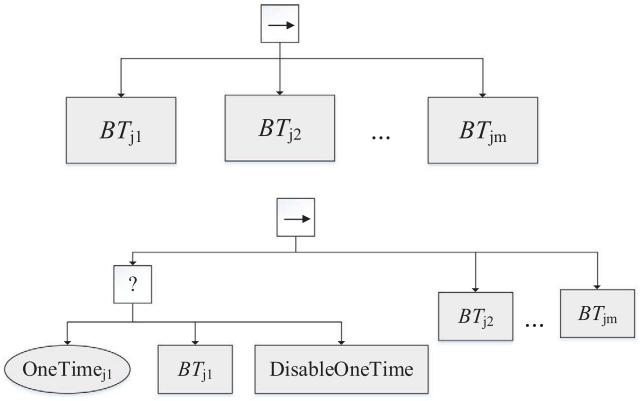


Fig. 3. Behavior tree BT_j which is formed by the sequence of BT_{jm}, $m = 1, \dots, M_j$, to meet the mission ϕ_j.

4.1. High-level behavior tree based planner synthesis

Here, we will discuss the process of generating a BT for a mission ϕ_j. Consider the F-LTL formula $\phi_j = \bigwedge_{m=1}^{M_j} \phi_{jm}$. Also, assume that there exist behavior trees BT_{jm}, $m = 1 \cdots M_j$, such that $BT_{jm} \vDash \phi_{jm}$ (we say $BT_{jm} \vDash \phi_{jm}$ if the execution of BT_{jm} satisfies ϕ_{jm}). Then, BT_j generated by the sequence of BT_{j1}, $\cdots$, BT_{jM_j}, as shown in Fig. 3, satisfies ϕ_j. Hence, the remaining problem is to synthesize BT_{jm}, $j = 1, \dots, M_j$, which are synthesized by Algorithm 2 (Lines 2–17). Note that we need to only synthesize sub-trees for sub-formulas of type $\Diamond C_{jm}$ and $\Box\Diamond C_{jm}$. This is due to the fact that specifications of types $\Box C_{jm}$ and $\Box(C'_{jm} \implies \bigcirc C_{jm})$ have to be always true, and hence, they do not need a BT, and instead they will be taken into account during the computation of the winning set W_j. Algorithm 2 implements this process. It starts by synthesizing sub-formulas of type $\Diamond C_{jm}$ (Lines 4–8), followed by synthesizing sub-formulas of type $\Box\Diamond C_{jm}$ (Lines 9–12). To create BT_j for the mission ϕ_j, each synthesized BTs are composed using a sequence node (Lines 7 and 11). Unlike the specification of type $\Box\Diamond C_{jm}$, the BT generated for specification of $\Diamond C_{jm}$ should be executed only one time, for which we have introduced global variables $OneTime_{jm}$, $m \in I_F$ that are initially set to $False$, and then, will be reset/disabled after execution of each sub-tree of type $\Diamond C_{jm}$ (see Line 6). Further, we introduced a mechanism to handle a conflict between BTs (Lines 13–16). For this purpose, if the last action of BT_{jm} violates the precondition of the first action of $BT_{j(m+1)}$, then there is a conflict. In order to resolve conflicts, the sequence of the BTs has to be switched, i.e., the priority of BT_{jm} has to be increased by shifting it to the left in the sequence node to be executed after $BT_{j(m+1)}$. To achieve this, the function `Conflict(.)` identifies the BT that is in conflict (Line 14) and the function `IncreasePriority(.)` (Line 15) increases the priority of BT_{jm}. This process is repeated until BT_j is conflict-free.

Algorithm 2: HighLevelBTSynthesis

1 **Function** HighLevelBTSynthesis(ϕ_j):
2 $onetime_{jm} \leftarrow False$ for $m \in I_{F_j}$
3 $BT_j \leftarrow \emptyset$
4 **for** $m \in I_{F_j}$ **do**
5 $BT_{jm} \leftarrow$ AddBTforSubFormula(ϕ_{jm})
6 $BT_{jm} \leftarrow$ Sequence(BT_j, $DisableOneTime$)
7 $BT_j \leftarrow$ Sequence(BT_j, BT_{jm})
8 **end**
9 **for** $m \in I_{AF_j}$ **do**
10 $BT_{jm} \leftarrow$ AddBTforSubFormula(ϕ_{jm})
11 $BT_j \leftarrow$ Sequence(BT_j, BT_{jm})
12 **end**
13 **do**
14 $conf\_flag, BT_{jm} \leftarrow$ Conflict(BT_j)
15 $BT_j \leftarrow$ IncreasePriority(BT_{jm})
16 **while** ($conf\_flag == True$)
17 **return** BT_j

Algorithm 3: AddBTforSubFormula

1 **Function** AddBTforSubFormula(ϕ_{jm}):
2 $BT_{jm} \leftarrow$ getcondtion(ϕ_{jm}) // set BT_{jm} to C_{jm}
3 ExpandBT(BT_{jm})
4 **if** $Type(C_{jm}) == Type(\Diamond C_{jm})$ **then**
5 $BT_{jm} \leftarrow$ Selector($OneTme_{jm}$, BT_{jm})
6 **end**
7 **return** BT_{jm}

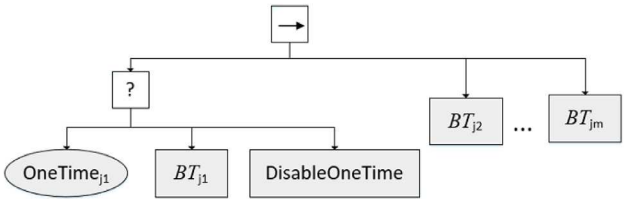


Fig. 4. BT_1 represents a specification of type $\Diamond C_{jm}$, followed by BTs for other types of formulas. The action $DisableOneTime$ disables the $OneTime$ flag to ensure that BT_1 will be executed only one time.

Called by Algorithm 2, Algorithm 3 initializes BT_{jm} with the condition C_{jm} for the sub-formula ϕ_{jm} (Line 2) and synthesizes the corresponding BT by calling the function ExpandBT(.) (Line 3). To enforce the execution of a sub-tree for type $\Diamond C_{jm}$ only once, BT_{jm} is composed with a boolean variable $onetime_{jm}$ by a *selector* node (Lines 4–6). The variable $OneTime_{jm}$ is initially set to $False$ to allow the execution of BT_{jm}, and then, the action $DisableOneTime$ sets $OneTime_{jm}$ to $True$ (Line 6 of Alg. 2) after the execution of that specific sub-tree for the first time. Since the composition is done by a selector node, setting $OneTime_{jm}$ to $True$ prevents subsequent execution of the corresponding sub-tree. Fig. 4 shows the resulting BTs following Algorithm 3.

Invoked by Algorithm 3, Algorithm 4 synthesizes BT's for each sub-formula's ϕ_{jm}. In a $While$ loop, by identifying the unmet conditions, BT_{jm} is modified in a recursive (backward) way until the synthesis of the BT is completed (Lines 2–8), i.e., the aforementioned backward process ends up with an action whose preconditions are already met (from which later we can start execution in a forward way). To implement this backward process, we use the function GetCondtionsToExpand(.) to pinpoint the cause, c_f (Line 3) and the function ExpandSubBT(.) to synthesize a suitable sub-tree to resolve the issue (Line 4). However, similar to the conflict between BT_{jm}'s, the addition of the new sub-tree may introduce a conflict that has to be resolved by adjusting the priory (Lines 5–7) of the newly added sub-tree. For example, if the goal is to close a door and there are two sub-trees: one with the action *closedoor* and the other with the action *opendoor*. The sub-tree with the action *opendoor* should come first, followed by the sub-tree with the action *closedoor*. If that is not the case, the priorities of the sub-trees should be adjusted.

Algorithm 4: ExpandBT

1 **Function** ExpandBT(BT_{jm}):
2 **do**
3 $c_f \leftarrow$ GetConditionToExpand(BT_{jm})
// Identify the cause for not being executable
4 $BT_{jm}, BT_{subtree} \leftarrow$ ExpandSubBT(BT_{jm}, c_f)
5 **while** $Conflict(BT_{jm})$ **do**
6 $BT_{jm} \leftarrow$ IncreasePriority($BT_{subtree}$)
7 **end**
8 **while** ($\neg IsSynthesisCompeleted(BT_{jm})$)

4.2. Computing winning regions for the system

Before executing the synthesized BTs for the mission ϕ_j, we should first compute the winning set W_j, which consists of states from where there exists a sequence of actions that satisfies the ϕ_j.

Algorithms 5–8 compute the winning sets for all types of sub-formulas whose intersection results in W_j.

Consider sub-formulas of type $\Box(C'_{jm} \implies \bigcirc C_{jm})$ in ϕ_j which require the robot to *respond* the current condition C'_{jm} by taking an action that meets C_{jm}. Putting all together, we form $\phi_{R_j} = \bigwedge_{i \in I_R} \Box(C'_{jm} \implies \bigcirc C_{jm})$.

Algorithm 5 computes the winning set, W_{R_j}, for ϕ_{R_j}. In Line 2, the variable W_{R_j} is set to S (the whole states of T_E), which is pruned by removing all states that are violating ϕ_{R_j} (Line 3–10). For each sub-formula, ϕ_{jm}, the algorithm computes $[\![C'_{jm}]\!]_{W_{R_j}} = \{q \in W_{R_j} \mid q \vDash C'_{jm}\}$, as the set of states that meet C'_{jm} (Line 4). Then, the algorithm removes those states from which there is no transition to a state that meets C_{jm} (Lines 5–9). After the termination of the loop, the states that satisfy ϕ_{R_j} are returned (Line 11).

Algorithm 5: Response

1 **Function** Response(ϕ_R):
2 $W_{R_j} \leftarrow S$
3 **for** $\phi_{jm} \in \phi_{R_j}$ **do**
4 $q \leftarrow [\![C'_{jm}]\!]_{W_{R_j}}$ // all states in which C'_{jm} can be met.
5 **for** $q_k \in q$ **do**
6 **if** $\forall a \in A{:}\delta(q_k, a) \notin [\![C_{jm}]\!]$ **then**
7 $W_{R_j} = W_{R_j} \,/\, q_k$ // remove state q_k from W_{R_j}
8 **end**
9 **end**
10 **end**
11 **return** W_{R_j}

Algorithm 6 computes W_{A_j} as the set of states that meet the safety requirement ϕ_{A_j}, where $\phi_{A_j} = \bigwedge_{i \in I_A} \Box C_{jm}$. In Line 2, the variable W_{A_j} is set to W_{R_j}, which was computed in Algorithm 5. The set W_{A_j} is then iteratively updated by removing states that are not safe (Lines 3–10). For each sub-formula, we set the variable S_{pre} to the set of states that satisfy C_{jm} (Line 4). Then, we compute $Cpred(S_{pre})$ which contains the states in which the agent can force a transition to a safe state under any environment action or non-determinism followed by updating S_{pre} by removing the states that are not in $Cpred(S_{pre})$ (Line 7). This process

is repeated until all the states can transition to an unsafe region are removed (Lines 5–8). In Line 9, the set W_{A_j} is updated by excluding unsafe states. At the end, the states that satisfy ϕ_{A_j} are returned (Line 11).

Algorithm 6: `Safety`

1 **Function** `Safety`(ϕ_{A_j}, W_{R_j}):
2 $W_{A_j} \leftarrow W_{R_j}$
3 **for** $\phi_{jm} \in \phi_{A_j}$ **do**
4 $S_{pre} \leftarrow [\![C_{jm}]\!]_{W_{A_j}}$
5 **do**
6 $temp = S_{pre}$
7 $S_{pre} = S_{pre} \cap \text{Cpre}(S_{pre})$
8 **while** $S_{pre} \neq temp$
9 $W_{A_j} \leftarrow W_{A_j} \cap S_{pre}$
10 **end**
11 **return** W_{A_j}

Next, we will compute the states that meet the specifications $\phi_{F_j} = \bigwedge_{i \in I_F} \Diamond C_{jm}$, and $\phi_{AF_j} = \bigwedge_{i \in I_{AF_j}} \Box\Diamond C_{jm}$. Unlike ϕ_{A_j} and ϕ_{R_j}, specifications ϕ_{F_j} and ϕ_{AF_j} have been taken into account during BT_j synthesis in Algorithm 2. Assuming that all sub-formulas of type $\Diamond C_{jm}$ or $\Box\Diamond C_{jm}$ describe reaching a destination, to compute the winning set for these specifications, we need to extract the $MoveTo()$ actions from BT_j by the function `MoveToExpand(.)` and sequence them according to the order that they will be executed by BT_j (Line 2 of Algorithm 7).

A similar procedure is followed to compute the reachable states for the action $MoveTo()$ that corresponds to specifications of type $\Diamond C_{jm}$ or $\Box\Diamond C_{jm}$. The only difference is that for $MoveTo()$ actions that correspond to the specifications of type $\Box\Diamond C_{jm}$, the robot should visit the target state(s) repeatedly. This difference can be captured by solving an additional reachability problem from the target states of the last $MoveTo()$ action to the target states of the first $MoveTo()$ action that corresponds to a specification of type $\Box\Diamond C_{jm}$. This is done by the function `ReturnToFirst(.)` which adds the return path to the sequence of $MoveTo()$ actions (Line 3 of Algorithm 7). Then, we compute the winning reachable states, W_{ju}, along with the reachable space, Γ_{ju}, for each subsequent $MoveTo(.)$ action target state by invoking Algorithm 8 (Lines 5–14). Initially s_{start} is set to s_0 (Line 7), while s_{next} is set to the target state of the first $MoveTo(.)$ action (Line 11). In subsequent iterations, s_{start} is set to the desired target state of the previous $MoveTo(.)$ action to keep track of the robot's position (Line 9), while s_{next} is set to the target state of the current $MoveTo(.)$ action (Line 11). At the mth iteration, we compute reachable states, W_{ju}, and reachable space $\Gamma_{ju}[m]$ for each $MoveTo(.)$ action (Line 12), where $\Gamma_{ju}[m]$ can be thought of as an array of reachable states for each single-step move (to be calculated in Algorithm 8). We then accumulate the reachable states for all $MoveTo(.)$ actions to calculate the winning reachable states (Line 13).

Algorithm 8 computes the states that are reachable from s_{next} in a backward way and verifies whether there is a viable path between s_{start} and s_{next}. For this purpose, we set $S_{cur} = \{s_{next}\}$ and then, using the operator $Cpre$, we iteratively update S_{cur} with one-step backward reachable states (Lines 5–13). In parallel, we also trace the states in each step by storing them in Γ. During this backward reachability computation, if we reach s_{start}, then the algorithm terminates returning S_{cur} and Γ (Lines 10–12). Otherwise, the algorithm continues until all reachable states are computed, after which if still $s_{start} \notin S_{cur}$, then the algorithm terminates returning $\emptyset$, which implies that there is no viable path between s_{start} and s_{next}.

Algorithm 9 computes the winning regions by invoking Algorithms 5–8. The set of allowable states, S, are pruned to meet specifications corresponding to ϕ_{R_j} (Lines 2), ϕ_{A_j} (Lines 3), as well as ϕ_{F_j} and ϕ_{AF_j} (Line 4). The synthesized BT is executable if the initial state of the robot is in the winning region returning W_j and Γ_{ju} (Lines 5–7). Otherwise BT_j is not executable and the algorithm returns $\emptyset$ (Lines 7–9).

Algorithm 7: `Reachable`

1 **Function** `Reachable`(BT_j, W_{A_j}):
2 $Move_{ju} = [\text{MoveToExpanded}(BT_j)]$
3 $Move_{ju_E} = [Move_{ju}, \text{ReturnToFirstAF}(BT_j)]$
4 $W_{ju} \leftarrow \emptyset$
5 **for** $m = 0$ *to* $m \leq |Move_{ju_E}|$) **do**
6 **if** *m==0* **then**
7 $s_{start} = s_0$
8 **else**
9 $s_{start} \leftarrow \text{GetTarget}(Move_{ju_E}[m-1], W_{A_j})$
10 **end**
11 $s_{next} \leftarrow \text{GetTarget}(Move_{ju_E}[m], W_{A_j})$
12 $W_{ji}, \Gamma_{ju}[m] \leftarrow \text{ReachablePath}(S_{start}, S_{next})$
13 $W_{ju} \leftarrow W_{ju} \cup W_{ji}$
14 **end**
15 **return** W_{ju}, Γ_{ju}

Algorithm 8: `ReachablePath`

1 **Function** `ReachablePath`$(s_{start}, s_{next}, W_{A_j})$:
2 $m = 0$
3 $S_{cur} = \{s_{next}\}$
4 $\Gamma[m] = \{s_{next}\}$
5 **do**
6 $m = m + 1$
7 $S_{prev} = S_{cur}$
8 $\Gamma[m] = Cpre(\Gamma[m-1])_{W_{A_j}}$
9 $S_{cur} = S_{cur} \cup \Gamma[m]$
10 **if** $s_{start} \in S_{cur}$ **then**
11 **return** S_{cur}, Γ
12 **end**
13 **while** $S_{prev} \neq S_{cur}$
14 **return** $\emptyset$

Algorithm 9: `GetWinningRegion`

1 **Function** `GetWinningRegion`(ϕ_j, BT_j):
2 $W_{R_j} \leftarrow S \cap \text{Response}(\phi_{R_j})$
3 $W_{A_j} \leftarrow S \cap \text{Safety}(\phi_{A_j}, W_{R_j})$
4 $W_j, \Gamma_{ju} \leftarrow \text{Reachable}(BT_j, W_{A_j})$
5 **if** $W_j \in s_o$ **then**
6 **return** W_j, Γ_{ju}
7 **else**
8 **return** $\emptyset$ `// unrealizable`
9 **end**

4.3. Executing BT

When the winning set, W_j, is not $NULL$, Algorithm 10 executes the generated BT, BT_j, to satisfy the mission ϕ_j (Lines 2–10). Since only the specifications of type $\Diamond C_{jm}$ and $\Box\Diamond C_{jm}$ are directly used to synthesize the BT, there are a total of $I = |I_F| + |I_{AF}|$ sub-BTs to execute. Therefore, to execute BT_j, in a *for-loop* (Lines 5–9), each sub-BT is executed to meet the conditions, C_{jm} (Lines 6–8). After executing all sub-BTs BT_{jm} of BT_j, the $OneTime_{jm}$ flags which correspond to the specification of type $\Diamond C_{jm}$ are disabled (the function `DisableOneTime()` in Algorithm 2 disables BT's specifications of type $\Diamond C_{jm}$), whereas the sub-BTs that correspond to the specifications of type $\Box\Diamond C_{jm}$ should continue executing until a new mission is introduced to the Robot. Here we assume that when a new mission is introduced, the previous

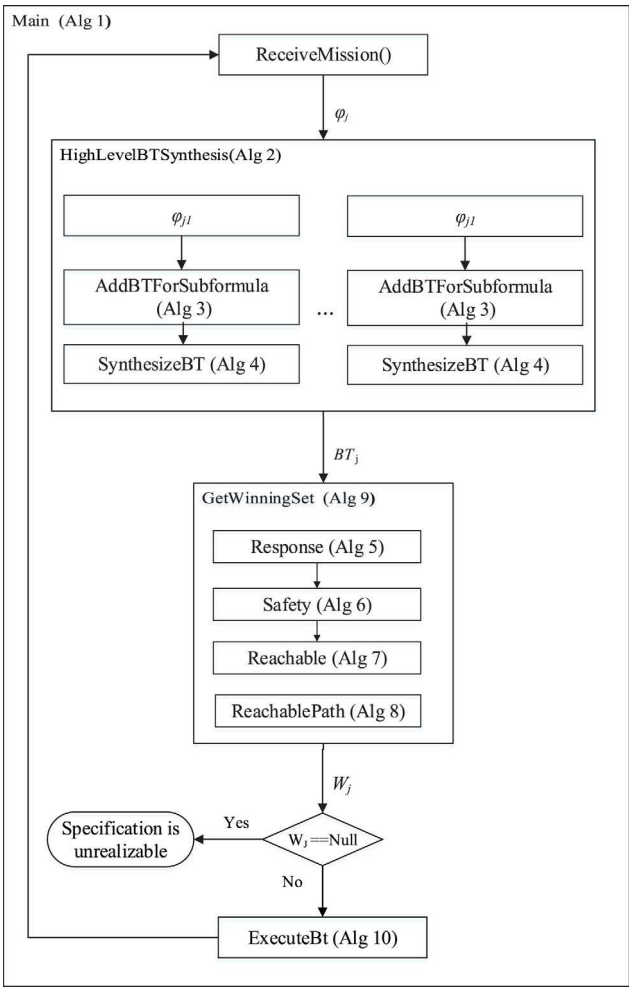


Fig. 5. Algorithms 1–10 describes the proposed framework for synthesis and execution of BTs for achieving missions given in the form of F-LTL specifications.

mission has to be terminated when all sub-BTs of the previous mission are executed at least once.

Algorithm 10: ExecuteBT

```
Function ExecuteBT(BT_j, Γ_ju):
    I_j = |I_Fj| + |I_AFj|
    Sucess_jm = False;  m = 1, ··· , I
    do
        for (m = 1 to m ≤ I_j) do
            while ¬Sucess_jm do
                Sucess_jm = Execute(BT_jm, Γ_ju)
            end
        end
    while (NoNewMission())
```

The overall flow of the algorithms is shown in Fig. 5. Upon receiving a mission, Algorithm 1, synthesizes a BT, computes its winning set and space, and finally executes the synthesized BT to satisfy the mission goals. Algorithm 2 synthesizes the BT for each mission ϕ_j by invoking Algorithms 3 and 4 to synthesize sub-BTs for specifications of type $\Diamond C_{jm}$ and $\Box\Diamond C_{jm}$. Based on the generated BT, Algorithm 9 computes the wining set, W_j, by invoking Algorithms 5–8. Then, if the winning set W_j is not null, Algorithm 10 executes the synthesized BT to meet the mission goals.

5. Properties of the proposed method

This section provides the proof of correctness and the analysis of the computational complexity of the proposed method.

5.1. Proof of correctness

Next, we show that the proposed method synthesizes a BT, B_j, and computes the winning set, W_j, to meet the mission goals for ϕ_j. Here, we assume that for each goal, there exists a sequence of actions that can be executed by the robot leading to the achievement of the goals of missions.

Lemma 1. *Algorithms 2–4 synthesize a BT that can satisfy the specifications of type $\Diamond C_{jm}$ and $\Box\Diamond C_{jm}$ for mission ϕ_j if $W_j \neq \emptyset$.*

Proof. Consider a specification of type $\Diamond C_{jm}$ or $\Box\Diamond C_{jm}$ in ϕ_j and assume that there exists a sequence of actions that meets C_{jm}, Algorithm 4 synthesizes a sub-BT to meet C_{jm}. Algorithm 3 expands the generated BT for C_{jm} to meet a specification of type $\Diamond C_{jm}$ by adding the flag $OneTime_{jm}$. Finally, Algorithm 2, sequences all sub-BTs for specifications of type $\Diamond C_{jm}$ and $\Box\Diamond C_{jm}$. Assuming that all actions are executable in finite time and $W_j \neq \emptyset$, by construction, executing the synthesized BT meets the goals $\Diamond C_{jm}$ and $\Box\Diamond C_{jm}$. □

Lemma 2. *Algorithm 6 terminates returning the set of states, W_{A_j}, that satisfies the goals for specification of type $\Box C_{jm}$.*

Proof. Algorithm 6 computes the set of states that meets the specification of type $\Box C_{jm}$. Here, we will prove the termination of the loop in Lines 5–8 in Algorithm 6 by showing that $Cpre(.)$ is monotone. Let, $S_1 \subseteq S_2$. Then $Cpre(S_2) = Cpre(S_1) \cup Cpre(S_2 \backslash S_1)$, implying that $Cpre(S_1) \subseteq Cpre(S_2)$. Thus $Cpre(.)$ is a monotonically increasing function. Since $Cpre(.)$ is a monotone function and the set, S, is finite, the loop for searching and removing states that do not meet $\Box C_{jm}$ terminates after a finite number of iterations. □

Lemma 3. *Algorithm 8 terminates returning all reachable states and paths between s_{start} and s_{next}.*

Proof. Algorithm 8 computes the set of states that are reachable from s_{next} and paths between s_{start} and s_{next} over the set S. Since $Cpre(.)$ is monotone (See Lemma 2 for the proof) and the set, S, is finite, the loop in Lines 5–13 of Algorithm 8 for finding the reachable states and paths terminates after a finite number of iterations. □

Theorem 1. *The proposed BT synthesis and execution approach, described in Algorithms 1–10, addresses the BT Synthesis and Execution Problem for F-LTL mission specifications, ϕ_j, if $W_j \neq \emptyset$.*

Proof. Algorithm 1 sequentially invokes Algorithms 2–4 for synthesis of the BT, Algorithms 5–9 for computation of winning set, and Algorithm 10 for the execution of the synthesized BT. According to Lemma 1, the synthesized BT by Algorithms 2-4 can meet the specification of type $\Diamond C_{jm}$ and $\Box\Diamond C_{jm}$ if $W_j \neq \emptyset$. On the other hand, Algorithms 5–9 compute the winning region, W_j that meets the F-LTL specification ϕ_j by a finite number of iterations of searches over a finite space (see Lemmas 2 and 3). Therefore, by construction, if $W_j \neq \emptyset$, execution of the synthesized BT by Algorithm 10 leads to the satisfaction of ϕ_j addressing the *BT Synthesis and Execution Problem* for F-LTL specification. □

Under this assumption, each mission goal can be achieved by a sequence of actions, we always can find a BT to complete the mission leading as stated in the following corollary.

Table 1
Action bank for the robot, R, where action $MoveTo(p)$ guides the robot R to position p, $Place(O,p)$ delivers an object O to a target location T, $Pick(O,p)$ picks an object O from position p, and $TakeImage(p)$ takes an image at location p.

Robot Action Bank			
Actions	Description	Precondition	Effect
A_1	MoveTo(p)	–	$R\ at\ p$
A_2	Pick(O,p)	$arm\ is\ free$ $R\ is\ near\ p$	$O\ is\ at\ R\ arm$
A_3	Place(O,T)	$R\ at\ T$ $O\ is\ at\ R\ arm$	$o\ at\ T$
A_4	TakeImage(p)	$R\ at\ p$	$TI\ at\ p$

Corollary 1. *Conducting Algorithms 1–9 enumeratively over all possible sequences of actions addresses the BT Synthesis and Execution Problem for F-LTL specification.*

5.2. Computational complexity analysis

The complexity analysis of the proposed framework can be broken down into the complexity of the BT synthesis module (Algorithms 2–4), the winning set computation module (Algorithms 5–9), and the BT execution module (Algorithm 10).

For the complexity analysis of the BT synthesis module, we start by considering Algorithm 4 that synthesizes a sub-BT, BT_{jm}, for a sub-formula, ϕ_{jm}. As shown in Algorithm 4, the synthesis stage starts by determining unmet conditions (Line 3 of Algorithm 4) and constructing the sub-tree that meets the condition (Line 4 of Algorithm 4). Assuming a lookup table containing the conditions and their corresponding actions with preconditions (the elements of the sub-tree) is available, the complexity for the synthesis of a sub-tree is $\mathcal{O}(1)$. Then, repeating this process for all unmet conditions and also handling possible conflicts in every iteration (Lines 5–7) which in the worst case need to be checked for conflict with $n-1$ sub-trees, where n is the maximum number of actions that are needed to complete a task, the complexity of Algorithm 4 is $\mathcal{O}(n^2)$. Since a mission is composed of totally $I_j = |I_{F_j}| + |I_{AF_j}|$ sub-BTs (Lines 4–12 of Algorithm 2), and each sub-BT has to be checked for possible conflicts with other sub-BTs, the synthesis module has a complexity of $\mathcal{O}\left(I_j^2 n^2\right)$.

The computational complexity of the winning set via Algorithms 5-9 is dominated by the computation of safe states (Algorithm 6) and reachable states (Algorithms 7 and 8). In Lemma 2, it is shown that the loop in Algorithm 6 (Lines 5–8) terminates after a finite number of iterations. As shown in Cormen et al. (2001), the computation of safe states by finding a fixed-point set via a depth-first search has a complexity of $\mathcal{O}(|S| + |\delta|)$, where $|S|$ is the number of states and $|\delta|$ is the total number of transitions. Thus, the complexity of Algorithm 6 is $\mathcal{O}\left(|I_{A_j}|(|S| + |\delta|)\right)$, where $|I_{A_j}|$ is the number of specifications of type $\square C_{jm}$. Similarly, the complexity of computing the reachable set via Algorithms 7 and 8 is $\mathcal{O}\left(|Move_{ju_E}|(|S| + |\delta|)\right)$, where $Move_{ju_E}$ is the number of $MoveTo(.)$ actions in BT_j.

Execution of each sub-tree in Lines 6–8 of Algorithm 10, has a complexity of $\mathcal{O}(n)$ where n is the maximum number of actions in a sub-tree. Since there are a total of I_j sub-BTs for a mission, ϕ_j, the complexity of BT execution module is $\mathcal{O}(|I_j|n)$.

By considering the contribution from each module, the proposed framework has a complexity of

$$\mathcal{O}\left(|Move_{ju_E}|(|S| + |\delta|) + |I_j|^2 n^2 + |I_j|n\right).$$

6. Simulation results

In this section, we consider two scenarios. First, we consider a robot R in a given environment to handle two missions, with the objective of illustrating the steps of the developed algorithms. In the second scenario, we apply the developed tasking algorithm to the coordination of the robot R for different sizes of environment and randomized places of obstacles and targets, to assess the runtime efficiency of the proposed method.

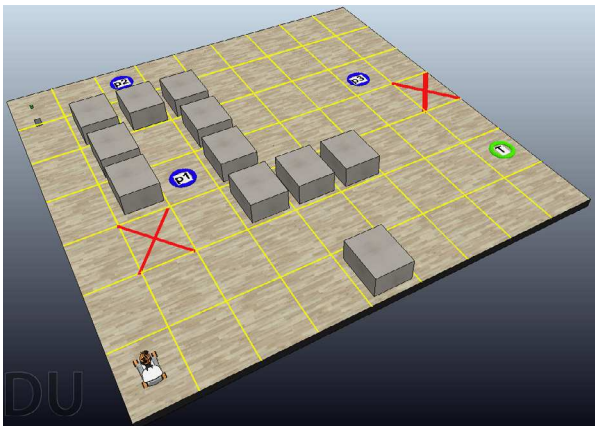


Fig. 6. Operational environment with 7×9 cells. The robot R can transit to its neighboring cells excluding occupied ones. The robot has to deliver object O to the target cell T, followed by persistent patrolling and taking pictures from targets P_1, P_2, and P_3, while avoiding the restricted zones RZ_1 and RZ_2.

6.1. Scenario 1

Consider an operational environment, shown in Fig. 6, which is partitioned into 7×9 grid cells, in which S_{ij} refers to the cell in ith row and jth column. There are also two restricted zones RZ_1 and RZ_2 located at S_{52} and S_{49}. Also, consider the robot R that starts from the initial position at S_{71}, and can transit to neighboring regions except the occupied cells. The robot R is assumed to be capable of executing actions A_1 ($MoveTo$), A_2 ($Pick$), A_3 ($Place$), and A_4 ($TakePicture$), whose associated preconditions and effects are listed in Table 1.

Now, consider two missions ϕ_1 and ϕ_2 that are introduced to the robot R sequentially for picking/placing an object and persistent patrolling, respectively. More specifically, missions ϕ_1, requires the robot R to pick an object O, located at S_{11}, and deliver it to the target cell T, located at S_{69}, while avoiding the restricted zones. This specification is captured as a F-LTL:

$$\phi_1 = \square\neg RZ_1 \wedge \square\neg RZ_2 \wedge \Diamond(O\ at\ T)$$

where RZ_1, RZ_2, and $O\ at\ T$ are atomic propositions that are true when the robot is at RZ_1, the robot is at RZ_2, and the Object is at target, respectively. During or after executing mission ϕ_1, mission ϕ_2 will be introduced with the objective of patrolling and taking pictures from targets p_1, p_2, and p_3 located at cells S_{43}, S_{13}, and S_{38}, respectively, while avoiding restricted zones, which is expressed as an F-LTL formula:

$$\phi_2 = \square\neg RZ_1 \wedge \square\neg RZ_2 \wedge \square\Diamond(TI\ at\ p_1) \wedge \square\Diamond(TI\ at\ p_2) \wedge \square\Diamond(TI\ at\ p_3)$$

where $TI\ at\ p_i$ is an atomic proposition that is true when the robot takes an image at position p_i.

Given the action bank, A, in Table 1, the process of generating BTs and computing W_j's, for missions ϕ_1 and ϕ_2 is as follows. The first mission, ϕ_1, is composed of three sub-formulas: $\phi_{11} = \square\neg RZ_1$, $\phi_{12} = \square\neg RZ_2$ and $\phi_{13} = \Diamond(O\ at\ T)$. Since the specifications of type $\square C_{jm}$ will be taken into account during the computation of the winning set and reachable space, we only need to synthesize $BT_1 = BT_{13}$ for $\phi_{13} = \Diamond(O\ at\ T)$. Invoked by Algorithm 1, Algorithm 2 initiates the synthesis of BT_{13} (Lines 4–8 of Algorithm 2) by calling Algorithm 3 to initialize BT_{13} with the condition $O\ at\ T$ (Line 2 of Algorithm 3) as shown in

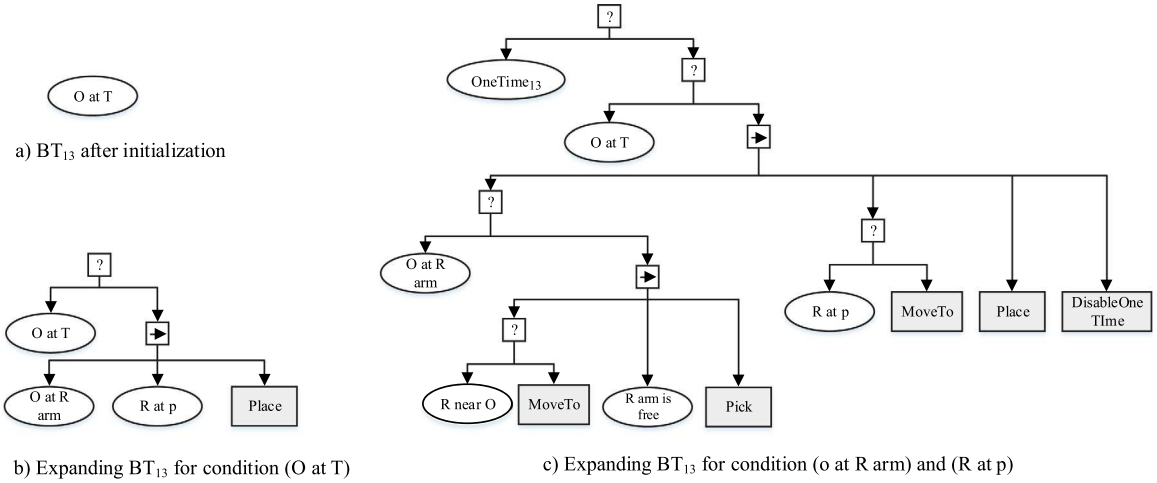


Fig. 7. Synthesizing BT_1 to deliver an object O to position T: (a) BT_{13} is initialized for specification $\Diamond(O\ at\ T)$, (b) The BT is expanded by identifying that the action "$Place$" can meet the condition $O\ at\ T$, and hence, the action "$Place$" and its preconditions ("$O\ at\ R\ arm$" and "$R\ at\ T$") are added to the BT, (c) The BT is expanded by identifying the actions and their preconditions for the conditions "$O\ at\ R\ arm$" and "$R\ at\ T$", as well as the action $DisbaleOneTime()$ to set the flag $oneTime_{13}$ after one time execution of BT_{13}.

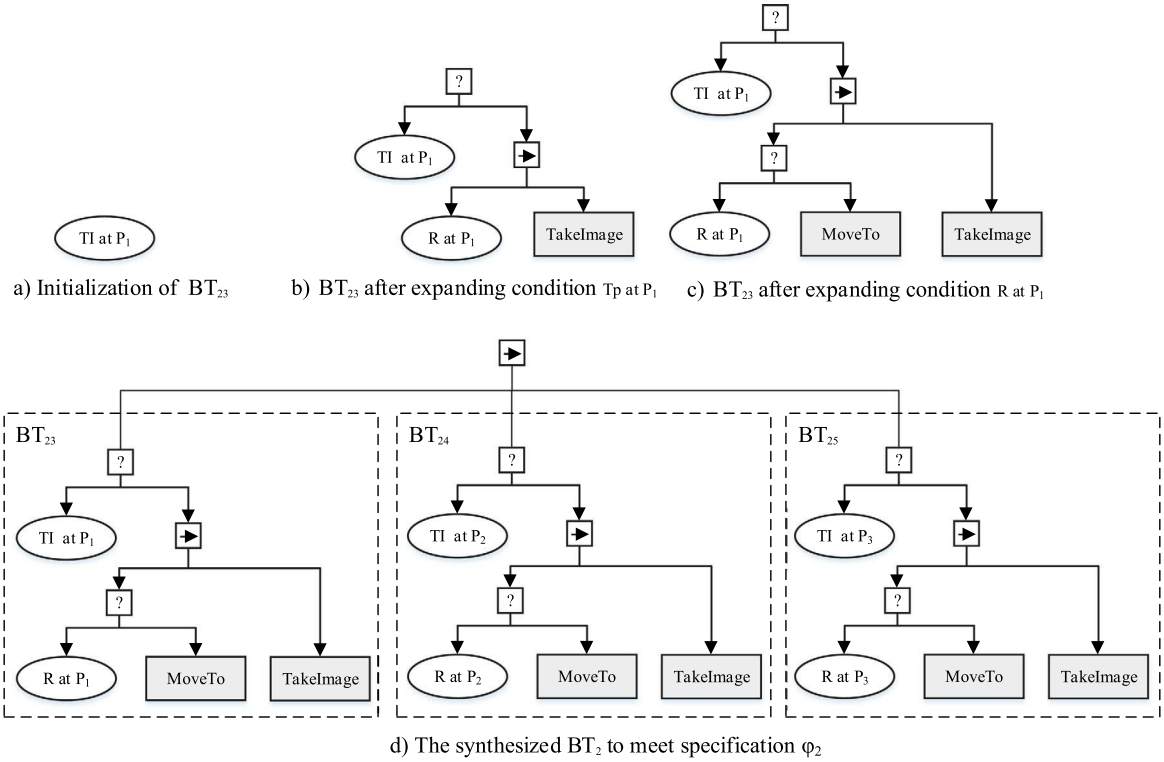
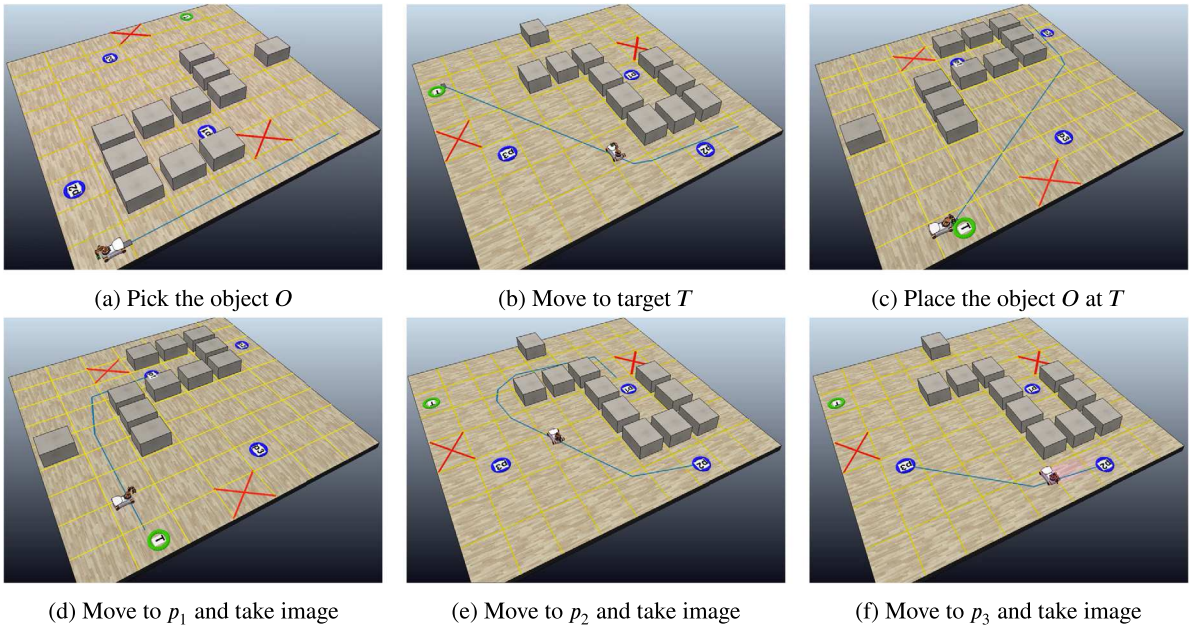


Fig. 8. Synthesizing BT_2 for ϕ_2 which requires patrolling positions p_1, p_2, and p_3 infinitely often while taking images: (a) BT_{23} is initialized for specification $\Diamond(TI\ at\ p_1)$, (b) The BT is expanded by identifying that the action "$TakeImage$" at p_1 can meet the condition "$TI\ at\ p_1$", and hence, the action "$TakeImage$" and its precondition $R\ at\ p_1$ are added to the BT, (c) The BT is expanded by identifying that the action "$MoveTo$" can meet the condition "$R\ at\ p_1$", and hence, the action "$MoveTo$" is added to the BT, (d) BT_2 is synthesized by combining BT_{23}, BT_{24}, and BT_{25} by a sequence node (for better visualization the $OneTime_{23}$ and $DisableOneTime$ is not shown for BT_{23}).

Fig. 7.a. Then, Algorithm 4 takes the initialized BT_{13} and continuously updates it by identifying an unmet condition and generating a sub-tree to meet the condition until the synthesis of BT_{13} is completed (it ends up with an action whose preconditions are already met). For this purpose, the function `GetCondtionsToExpand(.)` in Line 3 of Algorithm 4 identifies the condition "$O\ at\ T$" as unmet. This will be followed by calling the function `ExpandBt(.)`, which uses a selector node to compose the condition "$O\ at\ T$" with the sub-tree that contains the action $Place$, in which the action $Place$ is composed with its preconditions by a sequence node (see Table 1 for list of preconditions and Fig. 7.b for the synthesized sub-BT). This process continues by expanding the BT to find the actions and their preconditions for the conditions "$O\ at\ R\ arm$" and "$R\ at\ T$". Fig. 7.c shows the expanded sub-BT including the flag $oneTime_{13}$ and the action $DisbaleOneTime()$ that are needed to ensure onetime execution of the specification "$\Diamond O\ at\ T$".

We next compute the winning region W_1 for BT_1. To simplify the explanation of the process, consider the following sets $OCC = \{S_{22}, S_{23}, S_{24}, S_{32}, S_{34}, S_{42}, S_{44}, S_{54}, S_{55}, S_{56}, S_{75}\}$, $RZ = \{S_{52}, S_{49}\}$, $target = \{S_{69}\}$ which represents occupied cells, restricted zones, and the target, respectively. Invoked by Algorithm 9, Algorithm 5 returns $W_{R_1} = S$ as there is no specification of type $C'_{jm} \implies \bigcirc C_{jm}$. Then, Algorithm 6 prunes the set of states that are not safe and returns the



(a) Pick the object O (b) Move to target T (c) Place the object O at T

(d) Move to p_1 and take image (e) Move to p_2 and take image (f) Move to p_3 and take image

Fig. 9. Execution of BT_1 (a–c) and BT_2 (d–f).

states $W_{A_1} = S \setminus \{RZ\}$ as the states in RZ are the only unsafe regions. This is followed by the identification of reachable states and spaces for each $MoveTo()$ action (Line 4 of Algorithm 9). For this purpose, Algorithm 9 calls Algorithm 7 to compute W_{1u} and Γ_1 for $MoveTo(S_{11})$ actions from the initial position $s_0 = S_{71}$ and for $MoveTo(S_{69})$ actions from the S_{11} ending up with $W_{1u} = S \setminus \{occ, RZ\}$. Putting all together in Algorithm 9, we have $W_1 = W_{R_1} \cap W_{A_1} \cap W_{1u} = S \setminus \{occ, RZ\}$.

Following the computation of the winning set, since W_1 is not $NUll$, Algorithm 10 executes BT_1. Fig. 9 (a–c) shows the execution of BT_1 starting where the robot moves towards the object O, pick O, move to the target location, and, finally place the object.

Once ϕ_1 execution is completed, the second mission, ϕ_2, is introduced to the robot which has five sub-formulas: $\phi_{21} = \square\neg RZ_1$, $\phi_{22} = \square\neg RZ_2$, $\phi_{23} = \square\lozenge(TI\ at\ p_1)$, $\phi_{24} = \square\lozenge(TI\ at\ p_2)$, and $\phi_{25} = \square\lozenge(TI\ at\ p_3)$. Following similar procedures, we synthesize BT_{23} for sub-formula ϕ_{23}, by initializing it as "$TI\ at\ p_1$" as shown in Fig. 8.a, and then expanding it for the condition "$TI\ at\ p_1$" and its precondition "$R\ at\ p_1$", as shown in Fig. 8.b and Fig. 8.c, respectively. BT_{24} and BT_{25} can be synthesized in the same way. By combining these sub-BTs using a selector node, we synthesize BT_2 for ϕ_2 as shown in Fig. 8.d. Then, the winning set, $W_2 = S \setminus \{obs, RZ\}$, and Γ_2 are computed where Γ_2 provides the reachable space for the four $MoveTo()$ actions ($MoveTo(p_1)$ from T, $MoveTo(p_2)$ from p_1, $MoveTo(p_3)$ from p_2, and $MoveTo(p_1)$ from p_3. Finally, since the winning set W_2 is not $NULL$, Algorithm 10 executes the synthesized BT to meet the mission ϕ_2. Fig. 9 (d–h) shows the execution of BT_2 where the robot moves to p_1, p_2, and, p_3 one after the other and simultaneously taking image at each position.

6.2. Scenario 2

Consider an operational environment, which is partitioned into $n \times n$ grid cells. There are two restricted zones RZ_1 and RZ_2 which are located at two different cells S_1 and S_2. Also, consider the robot R that starts from the initial position at S_0, and can transit to neighboring regions except for the occupied cells. The robot R is assumed to be capable of executing actions A_1 ($MoveTo$), and A_2 ($TakePicture$), whose associated preconditions and effects are listed in Table 1. Now, consider the missions ϕ_3, which requires the robot R to conduct persistent patrolling. More specifically, mission ϕ_3 has the objective of patrolling and taking pictures at targets p_1 (eventually once) and p_2 (infinitely often), while avoiding restricted zones. The mission ϕ_3 can be captured as:

$$\phi_3 = \square\neg RZ_1 \wedge \square\neg RZ_2$$
$$\lozenge(TI\ at\ p_1) \wedge \square\lozenge(TI\ at\ p_2)$$

Table 2
Time taken for the BT synthesis, computation of the winning set, and execution of BT_3 for an $n \times n$ grid environment.

Size of the environment	BT Synthesis	Safety (Alg. 6)	Reachable (Algs. 7 and 8)	Execution (Alg. 10)
5 × 5	0.73 ms	0.227 ms	0.518 ms	9 s
10 × 10	0.73 ms	6.86 ms	16.9 ms	19 s
100 × 100	0.73 ms	23.7 ms	55.9 ms	199 s
1000 × 1000	0.73 ms	2546 ms	8098 ms	1999 s

We synthesize and execute the BTs for different sizes of the environment while randomizing the placement of the restricted zones RZ_1 and RZ_2, the robot initial position S_0, and the target locations p_1, and p_2.

Given the action bank, A, in Table 1, for each setup, we apply the developed algorithms to generate BTs and computing W_j's, for missions ϕ_3, which has four sub-formulas: $\phi_{31} = \square\neg RZ_1$, $\phi_{32} = \square\neg RZ_2$, $\phi_{33} = \lozenge(TI\ at\ p_1)$, and $\phi_{34} = \square\lozenge(TI\ at\ p_2)$. Following a procedure similar to Scenario 6.1, the BTs for ϕ_{33} and ϕ_{34} are synthesized and then composed using a selector node. The synthesized BT_3 for ϕ_3 is shown in Fig. 10.

Then, the winning set, $W_3 = S \setminus \{RZ\}$, and Γ_3 are computed, where Γ_3 provides the reachable space for the two $MoveTo()$ actions: $MoveTo(p_1)$ for moving from s_0 to p_1, and $MoveTo(p_2)$ for moving from p_1 to p_2. Since the winning set W_3 is not $NULL$, Algorithm 10 executes the synthesized BT to meet the mission ϕ_3.

The synthesized BT is the same for different configurations of the environment and the location of the initial position of the robot, restricted zones, and targets. However, for different configurations, the computation time of the winning set and the execution time of the BT are different as the computation of safes states (Algorithm 6) and reachable states (Algorithms 7 and 8), and the execution of the BT in Algorithm 10 depend on the locations of the robot's initial position, restricted zones, and targets. Table 2, provides average time for synthesis of the BT, the computation of the winning set and execution of the BT.

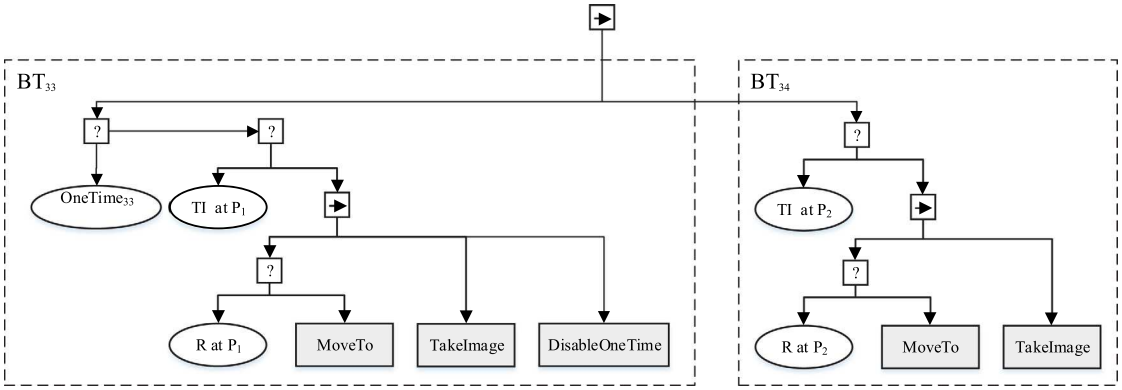


Fig. 10. BT_3 is synthesized by combining BT_{33}, and BT_{34} by a sequence node.

The execution time of the BT is obtained by assuming the robot moves from one cell to any of its neighbor cells in 1 s.

7. Conclusion

This paper developed an automatic provably-correct online BT synthesis and execution technique for the coordination of an autonomous system to accomplish a series of missions which are introduced to the system on-the-fly. Capturing the mission requirements in the form of F-LTL formulas, we developed a novel top-down, divide-and- conquer approach to decompose the missions into smaller sub-formulas, for which we designed sub-BTs. The realizability of the sub-BTs was checked by computing the intersection of safe and reachable sets in parallel to storing the paths to the winning set. If realizable, these sub-BTs are composed in order to form a coordinator to achieve the assigned mission by executing the synthesized BT using the calculated paths to the winning set. The correctness of the proposed method was proved. Unlike many existing methods which rely on manually designing the BTs, our proposed method can automatically synthesize a BT for a given F-LTL specification. Further, compared with most existing results which suffer from exponential complexity, we proved that the complexity of the proposed method is polynomial in the size of the formula and the size of the environment. The developed method was applied to the case-studies for different missions and different sizes of the environment using a physics-based simulator, demonstrating the capability of the proposed method on handling complex missions and scalability of the approach in terms of the size of the environment.

CRediT authorship contribution statement

Tadewos G. Tadewos: Methodology, Conceptualization, Software, Writing – original draft. **Abdullah Al Redwan Newaz:** Writing – review & editing, Software. **Ali Karimoddini:** Supervision, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The authors would like to acknowledge the support from National Science Foundation under the award number 1832110, Air Force Research Laboratory and OSD under agreement number FA8750-15-2-0116, and NASA University Leadership Initiative (ULI) under Grant 80NSSC20M0161.

References

Agis, R. A., Gottifredi, S., & García, A. J. (2020). An event-driven behavior trees extension to facilitate non-player multi-agent coordination in video games. *Expert Systems with Applications*, *155*, Article 113457.

Camacho, A., Baier, J. A., Muise, C., & McIlraith, S. A. (2018). Synthesizing controllers: On the correspondence between LTL synthesis and non-deterministic planning. In *Canadian conference on artificial intelligence* (pp. 45–59). Springer.

Colledanchise, M., Murray, R. M., & Ögren, P. (2017). Synthesis of correct-by-construction behavior trees. In *2017 IEEE/RSJ international conference on intelligent robots and systems* (pp. 6039–6046). IEEE.

Colledanchise, M., & Ögren, P. (2017). How behavior trees modularize hybrid control systems and generalize sequential behavior compositions, the subsumption architecture, and decision trees. *IEEE Transactions on Robotics*, *33*(2), 372–389.

Colledanchise, M., Parasuraman, R., & Ögren, P. (2019). Learning of behavior trees for autonomous agents. *IEEE Transactions on Games*, *11*(2), 183–189.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2001). Introduction to algorithms, (2nd ed.). The MIT Press.

Coronado, E., Indurkhya, X., & Venture, G. (2019). Robots meet children, development of semi-autonomous control systems for children-robot interaction in the wild. In *2019 IEEE 4th international conference on advanced robotics and mechatronics* (pp. 360–365). IEEE.

Dey, R., & Child, C. (2013). Ql-bt: Enhancing behaviour tree design and implementation with q-learning. In *2013 IEEE conference on computational inteligence in games* (pp. 1–8). IEEE.

Fainekos, G. E. (2011). Revising temporal logic specifications for motion planning. In *2011 IEEE international conference on robotics and automation* (pp. 40–45). IEEE.

Feng, L., & Wonham, W. M. (2008). Supervisory control architecture for discrete-event systems. *IEEE Transactions on Automatic Control*, *53*(6), 1449–1461.

French, K., Wu, S., Pan, T., Zhou, Z., & Jenkins, O. C. (2019). Learning behavior trees from demonstration. In *2019 international conference on robotics and automation* (pp. 7791–7797). IEEE.

Guerin, K. R., Lea, C., Paxton, C., & Hager, G. D. (2015). A framework for end-user instruction of a robot assistant for manufacturing. In *2015 IEEE international conference on robotics and automation* (pp. 6167–6174).

Guo, M., Johansson, K. H., & Dimarogonas, D. V. (2013). Revising motion planning under linear temporal logic specifications in partially known workspaces. In *2013 IEEE international conference on robotics and automation* (pp. 5025–5032). IEEE.

Iovino, M., Scukins, E., Styrud, J., Ögren, P., & Smith, C. (2020). A survey of behavior trees in robotics and ai. arXiv preprint arXiv:2005.05842.

Kim, S.-K., Myers, T., Wendland, M.-F., & Lindsay, P. A. (2012). Execution of natural language requirements using state machines synthesised from behavior trees. *Journal of Systems and Software*, *85*(11), 2652–2664.
Kloetzer, M., & Mahulea, C. (2015). LTL-based planning in environments with probabilistic observations. *IEEE Transactions on Automation Science and Engineering*, *12*(4), 1407–1420.
Lee, D., & Yannakakis, M. (1996). Principles and methods of testing finite state machines-a survey. *Proceedings of the IEEE*, *84*(8), 1090–1123.
Lin, S.-W., & Hsiung, P.-A. (2011). Counterexample-guided assume-guarantee synthesis through learning. *IEEE Transactions on Computers*, *60*(5), 734–750.
Maoz, S., & Shevrin, I. (2020). Just-in-time reactive synthesis. In *2020 35th IEEE/ACM international conference on automated software engineering* (pp. 635–646).
Nicolau, M., Perez-Liebana, D., O'Neill, M., & Brabazon, A. (2017). Evolutionary behavior tree approaches for navigating platform games. *IEEE Transactions on Computational Intelligence and AI in Games*, *9*(3), 227–238.
Pnueli, A., & Rosner, R. (1989). On the synthesis of a reactive module. In *Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on principles of programming languages* (pp. 179–190). ACM.
Raman, V., Donzé, A., Sadigh, D., Murray, R. M., & Seshia, S. A. (2015). Reactive synthesis from signal temporal logic specifications. In *Proceedings of the 18th international conference on hybrid systems: computation and control* (pp. 239–248).
Ruifeng, L., Jiasheng, W., Haolong, Z., & Mengfan, T. (2019). Research progress and application of behavior tree technology. In *2019 6th international conference on behavioral, economic and socio-cultural computing* (pp. 1–4).
Scheper, K. Y., Tijmons, S., de Visser, C. C., & de Croon, G. C. (2016). Behavior trees for evolutionary robotics. *Artificial life*, *22*(1), 23–48.
Shamgah, L., Tadewos, T. G., Karimoddini, A., & Homaifar, A. (2018). Path planning and control of autonomous vehicles in dynamic reach-avoid scenarios. In *2018 IEEE conference on control technology and applications* (pp. 88–93).
Tadewos, T. G., Shamgah, L., & Karimoddini, A. (2019a). Automatic safe behaviour tree synthesis for autonomous agents. In *Proc. of 58th IEEE conference on decision and control*.
Tadewos, T. G., Shamgah, L., & Karimoddini, A. (2019b). On-the-fly decentralized tasking of autonomous vehicles. In *Proc. of 58th IEEE conference on decision and control*.
Tumova, J., Marzinotto, A., Dimarogonas, D. V., & Kragic, D. (2014). Maximally satisfying LTL action planning. In *2014 IEEE/RSJ international conference on intelligent robots and systems* (pp. 1503–1510).
Wang, X., Moor, T., & Li, Z. (2020). Top-down nested supervisory control of state-tree structures based on state aggregations. *IFAC-PapersOnLine*, *53*(2), 11175–11180, 21st IFAC World Congress.
Wang, D., Wang, X., & Li, Z. (2020). Nonblocking supervisory control of state-tree structures with conditional-preemption matrices. *IEEE Transactions on Industrial Informatics*, *16*(6), 3744–3756.
Wolff, E. M., Topcu, U., & Murray, R. M. (2013). Efficient reactive controller synthesis for a fragment of linear temporal logic. In *2013 IEEE international conference on robotics and automation* (pp. 5033–5040). IEEE.
Wongpiromsarn, T., Topcu, U., & Murray, R. M. (2012). Receding horizon temporal logic planning. *IEEE Transactions on Automatic Control*, *57*(11), 2817–2830.
Wu, B., Dai, J., & Lin, H. (2015). Combined top-down and bottom-up approach to cooperative distributed multi-agent control with connectivity constraints. *IFAC-PapersOnLine*, *48*(27), 224–229, Analysis and Design of Hybrid Systems ADHS.
Wu, B., & Lin, H. (2016). Counterexample-guided distributed permissive supervisor synthesis for probabilistic multi-agent systems through learning. In *2016 American control conference* (pp. 5519–5524).

Tadewos G. Tadewos received his bachelor's in Computer and Electrical Engineering from Addis Ababa University in 2009. He later received his M.Sc in Embedded Systems from Polytecnico di Torino in 2014. In 2016, he joined the North Carolina A&T State University to pursue his Ph.D. He is a member of the Autonomous Cooperative Control of Emergent System of Systems (ACCESS) Laboratory and his research interests include Testing and Evaluation of Autonomous Vehicles, Formal Verification, tasking of multi-agent systems, and Aerial and Ground Robotics.

Abdullah Al Redwan Newaz is a Postdoctoral Research Associate at North Carolina A&T State University. Before that, he was a Postdoctoral Researcher at Rice University, Houston, TX, USA, and Nagoya University, Nagoya, Japan, in 2018-2020, 2017-2018, respectively. He earned Ph.D. and M.S. degrees in Information Science from Japan Advanced Institute of Science and Technology, Ishikawa, Japan, in 2017 and 2014, respectively. He received a B.Sc. degree in Mechanical Engineering from Rajshahi University of Engineering and Technology, Rajshahi, Bangladesh, in 2011. His research interests include autonomous systems, applied machine learning, motion planning under uncertainty, optimal control, deep learning-based perception methods, model checking, and related domains. He is an IEEE professional member and serves as an active reviewer for robotics, control, and transportation conferences and journals.

Ali Karimoddini is the Director of NC Transportation Center of Excellence on Connected andAutonomous Vehicle Technology (NC-CAV), the Director of the Autonomous Cooperative Control of Emergent System of Systems (ACCESS) Laboratory, and the Deputy Director of the TECHLAV DoD Center of Excellence on Autonomy. He received his Bachelor of Electrical and Electronics Engineering from the Amirkabir University of Technology in 2003 and Master of Science in Instrumentation and Automation Engineering from Petroleum University of Technology, in 2007. In 2008, he joined the National University of Singapore to pursue his Ph.D. degree. He is currently an Associate Professor at the Department of Electrical and Computer Engineering, North Carolina A&T State University. His research interests include Testing, evaluation, and control of autonomous vehicles, Cyber-physical systems, Cooperative control of multi-agent systems, Distributed decision making, Reliable, secure, and fault tolerant systems, and Human-machine interactions.