Synthesis of Correct-by-Construction Behavior Trees

Michele Colledanchise[†], Richard M. Murray[‡], and Petter Ögren[†]

***Abstract*— In this paper we study the problem of synthesizing correct-by-construction Behavior Trees (BTs) controlling agents in adversarial environments. The proposed approach combines the modularity and reactivity of BTs with the formal guarantees of Linear Temporal Logic (LTL) methods. Given a set of admissible environment specifications, an agent model in form of a Finite Transition System and the desired task in form of an LTL formula, we synthesize a BT in polynomial time, that is guaranteed to correctly execute the desired task. To illustrate the approach, we present three examples of increasing complexity.**

I. Introduction

As Behavior Trees (BTs) receive an increasing amount of attention, not only in computer game AI design textbooks [1], [2], but also in robotics [3]–[5], the lack of formal design methods is becoming a problem. The difficulty of analyzing and debugging BTs, as described in [6], lies in the particular execution of the BTs, which is less intuitive that a more classical Finite State Machine (FSM). Linear Temporal Logic (LTL) has proved to be a successful tool in addressing such problems in the hybrid control community, hence in this paper we propose to use LTL to synthesize correct-by-construction BTs given the system model and some assumptions on the environment.

Motivated by the aforementioned problems on analyzing and debugging BTs, we propose an approach that automatically synthesizes a BT that *guarantees* that a given agent will complete a given task, under some assumptions on the agent, task and environment. This is done in polynomial time, without compromising the advantages of BTs, in terms of reactiveness, modularity and human readability.

BTs are a control architecture first introduced in the video game industry to control in-game opponents, and are now an established tool appearing in textbooks [2] and generic game-coding software. BTs are appreciated for being highly modular, flexible and reusable, and have also been shown to generalize other successful control architectures such as the Subsumption architecture [7] and the Teleo-reactive Paradigm [8]. BTs have been used in applications including unmanned aerial vehicles [3], medical robotics [5], industrial robotics [4], and AI [9], [10]. In these applications, BTs are either manually designed by human experts or automatically designed using machine learning techniques [9] defining an objective function to maximize using heuristic methods.

LTL is a specification language [11] that allows a formal description of system properties that change over time and thus specify a wide variety of tasks, such as safety, response, persistence, and recurrence. LTL-based planning allows the automatic synthesis of correct-by-construction control policies [12], [13]. In an LTL-based planner, usually the task is specified in terms of an LTL formula with respect to a finite transition system modeling the system. The computational complexity of synthesizing a control policy that satisfies an LTL is doubly-exponential in the formula length [14], which led to the interest in identifying a fragment of LTL that is sufficiently expressive and for which, in the context of timed automata, the synthesis of a control policy is efficient. A common fragment used is the Generalized Reactivity (1) (GR(1)) [14]. LTL-planning is often used considering the presence of an adversarial environment. Then, the control policy synthesis assumes some behavior of the environment in form of a LTL formula. Off-the-shelf planners, such as [12], [13], receive as input the system model, assumption on admissible environment, and the system specification. To account for the adversarial environment, different techniques have been proposed: two players games, where the environment is considered as a player in the game [15]; the entire system as a non deterministic transition system where the environment decides which is the post-state for each action [16]; and allowing the environment to arbitrary change the robot state at a finite number of time instants [17].

We combine BTs with LTL by the automatic synthesis of a control policy in form of a BT that is *guaranteed* to satisfy a task defined in form of an LTL formula, under certain assumptions on the environment. This synthesis is done in polynomial time, with respect to the size of the system. In detail, we define the system model as a *finite transition system*, and synthesize a correct-by-construction BT that describes the policy that satisfies the desired task. Using the structure of BTs, the proposed controller is robust, in the sense of action failure handling, and reactive, in the sense of being able to countermeasure the environment that arbitrarily changes the robot state as in [17]. BTs do not limit the expressible control policy, as a BT is as expressive as a FSM and vice-versa [18].

To synthesize the BT in polynomial time we focus our attention on a fragment of LTL, similar to the fragment used in [16], which can specify tasks such as as safe navigation, response to the environment, persistent coverage, surveillance, guarantee and obligation.

The paper is organized as follows: Section II presents the related work. Section III describes BTs and LTL. Section IV formulates the problem and Section V the proposed approach. Section VI provides the theoretical analysis. Section VII shows the results. Section VIII concludes the paper.

[†]Robotics, Perception and Learning Lab, The Royal Institute of Technology, Stockholm, Sweden. e-mail: {miccol|petter}@kth.se

[‡]Department of Control and Dynamical Systems, California Institute of Technology, Pasadena, CA, USA. e-mail: murray@cds.caltech.edu

II. Related Work

In this section we briefly summarize related work and compare it with the proposed approach.

The vast majority of BTs are still manually designed by human experts [1], [2], however, there has been a number of efforts to automate the process [9], [10]. In particular [9] proposes a grammar-based genetic programming method that combines a set of pre-defined sub-BTs to maximize a given objective function in closed form. The approach works well for the kind of applications where the task is to reach a given state and the objective function can be easily derived. In [19] the authors combine BTs with Q-learning, proposing an automated tree design based on reinforcement learning techniques. However, these approaches are applicable where the defined task is to satisfy a single proposition (e.g. reach a certain state, satisfy a given condition). In contrast, our approach considers a larger set of constraints.

Similar works [10] use machine learning techniques to learn a BT given a reward function. However this heuristic approach has no theoretical guarantee that the task will be correctly completed. In contrast, our approach provides guarantees that the desired task will be completed. Similar solutions have been proposed in the field of robotics, where genetic programming is applied directly onto the BT. Their approach involved experiments with a flying robot, iteratively learning the BT in every experiment. An attempt to find a safe behavior is found in [20] where they exploit the particular structure and execution of BTs to learn a *safe* BT. However both approaches are still based on heuristic functions that cannot capture behaviors that are easily described by LTL, such as surveillance and persistent coverage.

Formal verification of BTs has been studied in [3] where they translate the BT into a tractable formalism. The formalism used is the *Attributive Language with Complements and concrete Domains ALC(D)*. However such verifications can be done only after designing the tree, which is supposedly done by a human expert.

The first approach to combine LTL planning with BTs is found in [21] where they synthesize a maximally satisfying control policy taking into account robot failures. However BTs are used only as a bridge between their task execution framework and the low level controllers of the robot. In contrast, our approach synthesizes the BT as a task execution framework, preserving the advantages of BTs in term of modularity, reactiveness, robustness and human readability. In out approach, the BT itself is directly synthesized from the LTL.

III. Background: BT and LTL

In this section we briefly describe BTs and LTL. A more detailed description of BTs can be found in [2], while more detailed description of LTL can be found in [22].

Behavior Trees: A BT is a graphical modeling language and a representation for execution of actions based on conditions and observations in a system. A BT is a directed rooted tree where each node is either a control flow node or an execution node. With the common definitions of *parent* node and *child* node. The root is the single node without parents, whereas all other nodes have one parent. The control flow nodes have one or more children, and the execution nodes have no children. Graphically, the children of nodes are placed below it.

The execution of a BT begins from the root node. It sends *ticks* with a given frequency to its child. A tick is a signal that allows the execution of a child. When a parent sends a tick to a child, the execution of this is allowed. The child returns to the parent a status *running* if its execution has not finished yet, *success* if it has achieved its goal, or *failure* otherwise.

There are four types of control flow nodes: fallback; sequence; parallel; and decorator, and two execution nodes (action and condition). Below we describe the execution of the nodes used in this paper.

The *fallback node* ticks its children from the left, returning success (running) as soon as it finds a child that returns success (running). It returns failure only if all the children return failure. When a child returns running or success, the fallback node does not tick the next child (if any). The fallback node is graphically represented by a box with a "?".

The *sequence node* ticks its children from the left, returning failure (running) as soon as it finds a child that returns failure (running). It returns success only if all the children return success. When a child returns running or failure, the sequence node does not tick the next child (if any). The sequence node is graphically represented by a box with a "$\rightarrow$".

The *action node* performs an action, returning success if the action is completed and failure if the action cannot be completed. Otherwise it returns running.

The *condition node* checks whether a condition is satisfied or not, returning success or failure accordingly. The condition node never returns running.

There also exists a non-reactive version of the control flow nodes described above, where the nodes remember which child has returned *success* or *failure*. Nodes with memory always tick the same child until this returns *success* or *failure*, ignoring its status after that. Such control flow nodes are often called *nodes with memory* and they are graphically represented with the addition of the symbol "$*$" (e.g. a sequence node with memory is graphically represented by a box with a "$\rightarrow *$").

Linear Temporal Logic: LTL is a powerful language that can be used to specify a wide range of important system behavior. A LTL formula is defined according to the following grammar:

$$\varphi \triangleq p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \bigcirc\varphi \mid \varphi_1 \mathcal{U} \varphi_2 \tag{1}$$

An LTL formula consists of a set of atomic propositions; Boolean operators and temporal operators. Here, $\neg$ is the Boolean *negation* operator; $\wedge$ is the Boolean *conjunction* operator; $\bigcirc$ is the temporal *next* operator; and $\mathcal{U}$ is the temporal *until* operator. Further operators such as disjunction ($\vee$); implication ($\Rightarrow$); eventually ($\Diamond$), always ($\Box$); infinitely often ($\Box\Diamond$); and eventually forever ($\Diamond\Box$) can be derived.

IV. Problem Formulation

In this section we give a set of assumptions and definitions and then we state the main problem.

Definition 1: A *finite transition system* is a tuple $\mathcal{T} = \langle S, A, \tau, s_0, AP, \lambda \rangle$ where S is a finite set of states, A is a finite set of actions, $\tau : S \times A \to S$ is a transition function, s_0 is an initial state, AP is a set of atomic propositions, and $\lambda : S \to 2^{AP}$ is a labeling function.

Definition 2: The *action function* $\alpha : S \to 2^A$ gives the set of available actions at a given state.

Definition 3: The *null action* ε is an action such that $s = \tau(s, \varepsilon) \ \forall s \in S$.

Definition 4: A state $s' \in S$ is said *successor state* of $s \in S$ if and only if $\exists a \in \alpha(s) : s' = \tau(s, a)$. The set of successor states of s is denoted by $P_{ost}(s)$.

Definition 5: A *run* $\sigma = s_0, s_1, s_2, \dots$ of a finite transition system is an infinite sequence of its states where s_i is the state at index i.

Definition 6: A state s satisfies a proposition p if and only if $\sigma \vDash p$.

Definition 7: A run σ is a *satisfying run* of an LTL formula φ if and only if $s_i \vDash \varphi \ \forall s_i \in \sigma$.

Definition 8: A *policy* is a map $\pi : S \times M \times 2^{AP} \to A \times M$ where M is a set of memory variables and 2^{AP} is a set of all possible propositions.

Definition 9: A policy π is a *satisfying policy* for an LTL formula φ ($\pi \vDash \varphi$) if its run from s_0 is a satisfying run.

Definition 10: An *agent specification* φ describes the possible behaviors of the agent and is defined as:

$$\varphi = \varphi_\Box \wedge \varphi_\Diamond \wedge \varphi_{\Rightarrow\Diamond} \wedge \varphi_{\Rightarrow\Box\Diamond} \wedge \varphi_{\Rightarrow\bigcirc}. \tag{2}$$

where:

$\varphi_\Box \triangleq \bigwedge_i \Box p_{1i}$, $\quad \varphi_\Diamond \triangleq \bigwedge_i \Diamond p_{2i}$

$\varphi_{\Rightarrow\Diamond} \triangleq \bigwedge_i \Box(q_{3i} \Rightarrow \Diamond p_{3i})$, $\quad \varphi_{\Rightarrow\Box\Diamond} \triangleq \bigwedge_i \Box(q_{4i} \Rightarrow \Box\Diamond p_{4i})$

$\varphi_{\Rightarrow\Diamond\Box} \triangleq \bigwedge_i \Box(q_{5i} \Rightarrow \Diamond \Box\, p_{5i})$, $\quad \varphi_{\Rightarrow\bigcirc} \triangleq \bigwedge_i \Box(q_{6i} \Rightarrow \bigcirc p_{6i})$

p_{ij} is an atomic proposition the agent can control and q_{ji} is an atomic proposition the agent cannot control.

Remark 1: The agent specification φ defined in Definition 10 allows $\varphi_{\Rightarrow\Diamond\Box}$ which cannot be specified in CTL or $GR(1)$. However it cannot allow disjunctions of formulas as in GR(1) without increasing the synthesis complexity.

Definition 11: The *environment specification* ψ describes the possible behaviors of the environments and is defined as:

$$\psi = \psi_\Box \wedge \psi_{\Box\Diamond} \wedge \psi_{\Diamond\Box} \tag{3}$$

where: $\psi_\Box \triangleq \bigwedge_j \Box q_{1j}$, $\psi_{\Box\Diamond} \triangleq \bigwedge_j \Box\Diamond q_{2j}$, $\psi_{\Diamond\Box} \triangleq \bigwedge_j \Diamond \Box\, q_{3j}$.

Assumption 1: $\varepsilon \in \alpha(s) \forall s \in S$. That is, the null action is available for each state in S.

Assumption 2: The initial state does not violate φ.

Assumption 3: The environment does not force the agent to a state where no possible actions are available (not even the null action).

Assumption 4: For each proposition p_{ij} to be satisfied, the environment does not block all the paths to a state $s_{ij} \in S : s_{ij} \vDash p_{ij}$ forever.

Remark 2: We need to formulate Assumptions 3 and 4 as we have no assumptions of when the variables q_{ji} become *true*.

Problem 1: Given a finite transition system $\mathcal{T}$ under Assumption 1, an LTL formula φ in form of (2), derive a BT such that its policy π holds the following $\pi \vDash \psi \Rightarrow \varphi$.

V. Proposed Solution

In this section we describe the proposed approach. We begin with an informal description of the algorithms, then we state a few definitions needed to give a formal description.

The behavior described by φ can be seen as the composition of three sub-behaviors: a *transient behavior* ($\varphi_\Diamond$); a *steady state behavior* ($\varphi_{\Rightarrow\Diamond\Box}$, $\varphi_{\Rightarrow\Box\Diamond}$, and $\varphi_{\Rightarrow\Box\Diamond}$); and a *priority behavior* ($\varphi_{\Rightarrow\bigcirc}$ and $\varphi_\Box$). We create a BT that describes the transient behavior and a BT that describes a steady state behavior. We compose the aforementioned trees in a sequence composition. Both sub-trees execute actions that satisfy the priority behavior. The satisfying run can thus be divided into a *transient* run and a *steady state* run.

For each formula in φ we compute three functions: the *value function*, the *requirement function*, and the *constraint function*. The value function is used to identify the actions the agent can perform to satisfy a given proposition. The requirement function has two purposes: it is used to identify at each state the actions that do not violate φ and it is used to define assumptions on the environment's specification ψ. The constraint function is used to draw assumptions on the environment's specification ψ.

Definition 12: The set $\Sigma(s) \subseteq S$ is the *satisfaction set* of s. It contains all the states in any satisfying run for φ starting from s.

Definition 13: $V : S \times AP \Rightarrow \mathbb{N} \cup \infty$ is the *value function* for a state $s \in S$ and a proposition $p \in AP$. Each value function satisfies the optimality condition:

$$V(s, p) = \min_{a \in \alpha(s)} V(\tau(s, a), p) + 1. \tag{4}$$

Definition 14: $R : S \Rightarrow 2^{AP}$ is the *requirement function* of the state $s \in S$.

The requirement function is computed by Algorithm 1 in $O(|\varphi||\mathcal{T}|)$ time. $R(s)$ satisfies the following: $R(s) \implies R_\Box(s) \wedge R_\bigcirc(s) \wedge R_\Diamond(s) \wedge R_{\Rightarrow\Diamond}(s) \wedge R_{\Box\Diamond}(s) \wedge R_{\Diamond\Box}(s) \wedge \bigvee_{s' \in P_{ost}(s) \setminus \{s\}} R(s')$ where $R_\Box(s)$, $R_\bigcirc(s)$, $R_\Diamond(s)$, $R_{\Rightarrow\Diamond}(s)$, $R_{\Box\Diamond}(s)$, and $R_{\Diamond\Box}(s)$ are defined below.

Function $\mathbf{R_\Box(s)}$*:* $R_\Box(s)$ is a proposition used to identify the action to perform in order to satisfy $\varphi_\Box$. The agent can move to s only if $R_\Box(s)$ holds. It is defined as:

$$R_\Box(s) \triangleq \bigwedge_i s \vDash p_{1i} \tag{5}$$

Intuitively, the agent can move to s only if $\varphi_\Box = \bigwedge_i \Box p_{1i}$ is satisfied.

Function $\mathbf{R_\bigcirc(s)}$*:* $R_\bigcirc(s)$ is a proposition used to identify the action to perform in order to satisfy $\varphi_{\Rightarrow\bigcirc}$. The agent can move to s only if $R_\bigcirc(s)$ holds. It is defined as:

$$R_\bigcirc(s) \triangleq \bigcirc AT_s \Rightarrow \bigwedge_{i:q_{6i}} s \vDash p_{6i} \tag{6}$$

where AT_s is a proposition that holds if and only if the agent is at s. Intuitively, the agent can move to s only if

Algorithm 1: get_R(S, φ, $R_\Box$, $R_\bigcirc$, $R_\Diamond$, $R_{\Box\Diamond}$, $R_{\Diamond\Box}$)

$\Gamma \leftarrow \varnothing$
for $s \in S$ **do**
  $R'(s) = R_\Box(s) \wedge R_\bigcirc(s) \wedge R_\Diamond(s)$
  for $p_{3i} \in \varphi$ **do**
    if $s \vDash p_{p_3 i}$ **then**
      $R'(s) = R'(s) \wedge R_{\Rightarrow\Box}(s)$
      $\Gamma \leftarrow \Gamma \cup \{s\}$
  for $p_{4i} \in \varphi$ **do**
    if $s \vDash p_{p_4 i}$ **then**
      $R'(s) = R'(s) \wedge R_{\Box\Diamond}(s)$
      $\Gamma \leftarrow \Gamma \cup \{s\}$
  for $p_{5i} \in \varphi$ **do**
    if $s \vDash p_{p_5 i}$ **then**
      $R'(s) = R'(s) \wedge R_{\Diamond\Box}(s)$
      $\Gamma \leftarrow \Gamma \cup \{s\}$
do
  $\Delta \leftarrow \varnothing$
  for $s' : s' \in$ `Parents(`Γ`)` **do**
    $R'(s) \leftarrow R'(s) \vee R'(s')$
    $\Delta \leftarrow \Delta \cup \{s'\}$
  $\Gamma \leftarrow \Delta$
while $\Gamma \neq \varnothing$;
for $s \in S$ **do**
  $R(s) = R_\Box(s) \wedge R_\bigcirc(s) \wedge R_\Diamond(s) \wedge R_{\Rightarrow\Diamond}(s) \wedge R_{\Box\Diamond}(s) \wedge R_{\Diamond\Box}(s) \wedge R'(s)$
return R

$\varphi_{\Rightarrow\bigcirc} = \bigwedge_i \Box(q_{6i} \Rightarrow \bigcirc p_{6i})$ is satisfied. The proposition AT_s will be helpful to avoid having the agent blocked at the initial state because $\neg q_{6i}$ holds despite the initial state satisfying p_{6i}.

Function $\mathbf{R_\Diamond(s)}$*:* $R_\Diamond(s)$ is a proposition used to identify the action to perform in order to satisfy $\varphi_\Diamond$. The agent can move to s only if $R_\Diamond(s)$ holds. It is defined as:

$$R_\Diamond(s) \triangleq \bigwedge_i (p^*_{2i} \vee V(s, p_{2i}) \in \mathbb{N}) \tag{7}$$

where $p^*_{2i}$is an auxiliary variable initialized to $false$ and set to $true$ when $s \vDash p_{2i}$. Intuitively, the agent can move to s if each proposition p_{2i} was either satisfied in the past or it can be satisfied in finite time.

Function $\mathbf{R_{\Rightarrow\Diamond}(s)}$*:* $R_{\Rightarrow\Diamond}(s)$ is a proposition used to identify the action to perform in order to satisfy $\varphi_{\Rightarrow\Diamond}$. The agent can move to s only if $R_{\Rightarrow\Diamond}(s)$ holds. $R_{\Rightarrow\Diamond}(s)$ is defined as $R_{\Rightarrow\Diamond}(s) \triangleq \bigwedge_i R'_{\Rightarrow\Diamond}(s, p_{3i})$, where $R'_{\Rightarrow\Diamond}(s, p_{3i})$ is defined as:

$$R'_{\Rightarrow\Diamond}(s, p_{3i}) \triangleq \begin{cases} \Box\neg q_{3i} \vee p^*_{3i} \vee \bigwedge\limits_{j:s\nvDash p_{3j}} (\Box\neg q_{3j} \vee (V(s, p_{3j}) \in \mathbb{N} \wedge \\ \wedge \exists \tilde{s} : V(\tilde{s}, p_{3j}) = 0 \wedge V(s', p_{3i}) \in \mathbb{N})), \text{ if } s \vDash p_{3i} \\ \bigvee\limits_{s' \in P_{ost}(s)\setminus\{s\}} R'_{\Rightarrow\Diamond}(s', p_{3i}) \quad \text{otherwise} \end{cases} \tag{8}$$

where $p^*_{3i}$is an auxiliary variable initialized to $false$ and set to $true$ when $s \vDash p_{3i}$ and set to false when $\neg q_{3i}$ holds.

Function $\mathbf{R_{\Box\Diamond}(s)}$*:* $R_\Diamond(s)$ is a proposition used to identify the action to perform in order to satisfy $\varphi_{\Box\Diamond}$. The agent can move to s only if $R_{\Box\Diamond}(s)$ holds. $R_{\Box\Diamond}(s)$ is defined as $R_{\Box\Diamond}(s) \triangleq \bigwedge_i R'_{\Box\Diamond}(s, p_{4i})$, where $R'_{\Box\Diamond}(s, p_{4i})$ is defined as:

$$R'_{\Box\Diamond}(s, p_{4i}) \triangleq \begin{cases} \Box\neg q_{4i} \vee \bigwedge\limits_{j:s\nvDash p_{4j}} (\Box\neg q_{4j} \vee (V(s, p_{4j}) \in \mathbb{N} \wedge \\ \wedge \exists \tilde{s} : V(\tilde{s}, p_{4j}) = 0 \wedge V(s', p_{4i}) \in \mathbb{N})), \text{ if } s \vDash p_{4i} \\ \bigvee\limits_{s' \in P_{ost}(s)\setminus\{s\}} R'_{\Box\Diamond}(s', p_{4i}) \quad \text{otherwise} \end{cases} \tag{9}$$

Intuitively, the agent can move to s if for each p_{4i} the proposition $\Box(q_{4i} \Rightarrow \Box\Diamond p_{4i})$ is either trivially satisfied (i.e. $\Box\neg q_{4i}$ holds) or it is possible to satisfy p_{4i} in finite time and for all the other propositions $\Box(q_{4j} \Rightarrow \Box\Diamond p_{4j})$ that are not trivially satisfied it is possible to satisfy p_{4j} and then satisfy p_{4i} in finite time.

Function $\mathbf{R_{\Diamond\Box}(s)}$*:* The agent must satisfy $\varphi_{\Diamond\Box}$ in the steady state run. Whenever a proposition q_{5i} holds the states that satisfy the steady state run may change. To identify those states, for each p_{5i}, we create the finite transition system $\mathcal{T}_{p_{5i}}$ from $\mathcal{T}$ removing the states that do not satisfy p_{5i}. Then, using Equation(4), we compute the value functions $V_{p_{5i}}(s, p)$ on $\mathcal{T}_{p_{5i}}$. This value function allows us to compute the conditions under which there exists a satisfying run for the steady state.

$R_{\Diamond\Box}(s)$ is a proposition used to identify the action to perform in order to satisfy $\varphi_{\Diamond\Box}$. The agent can move to s only if $R_{\Diamond\Box}(s)$ holds. $R_{\Diamond\Box}(s)$ is defined as $R_{\Diamond\Box}(s) \triangleq \bigwedge_i R'_{\Diamond\Box}(s, p_{5i})$, where $R'_{\Diamond\Box}(s, p_{5i})$ is defined as:

$$R'_{\Diamond\Box}(s, p_{5i}) \triangleq \begin{cases} (\Diamond\,\Box\,\neg q_{5i} \vee \bigwedge\limits_{j:s\nvDash p_{5j}} \Diamond\,\Box\,\neg q_{5j}) \wedge (\Box\Diamond\neg q_{5i} \vee \\ \vee((\bigwedge_j \Diamond\,\Box\,\neg q_{4j} \vee \bigwedge_{k\neq j} \Diamond\,\Box\,\neg q_{4k} \vee (\exists s' : V_{p_{5i}}(s', p_{4k}) = 0 \wedge \\ \wedge V_{p_{5i}}(s', p_{4j}) \in \mathbb{N}) \wedge \bigwedge_j \Diamond\,\Box\,\neg q_{4j} \vee \bigwedge_{k\neq j} \Diamond\,\Box\,\neg q_{4k} \vee \\ \vee(\exists s' : V_{p_{5i}}(s', p_{4k}) = 0 \wedge V_{p_{5i}}(s', p_{4j}) \in \mathbb{N})))), \quad \text{if } s \vDash p_{5i} \\ \bigvee\limits_{s' \in P_{ost}(s)\setminus\{s\}} R'_{\Diamond\Box}(s', p_{5i}) \quad \text{otherwise} \end{cases} \tag{10}$$

Intuitively, the agent can move to s if for each p_{5i} the proposition $\Box(q_{5i} \Rightarrow \Diamond\,\Box\,p_{5i})$ is either trivially satisfied (i.e. $\Diamond\,\Box\,\neg q_{5i}$) or eventually forever does not have to reach a state (or a set of states) for which p_{5i} does not hold. Moreover eventually forever the agent must stay within states that satisfy p_{5i}; $\varphi_{\Rightarrow\Diamond}$; and $\varphi_{\Rightarrow\Box\Diamond}$

Definition 15 (Constraint Function): $\rho : S \Rightarrow 2^{AP}$ is the *constraint function* of the state s, defined as $\rho(s) = (\rho_{\Diamond\Box}(s) \wedge \bigvee_{s' \in P_{ost}(s)\setminus\{s\}} \rho_{\Diamond\Box}(s')) \wedge (\rho_\bigcirc(s) \wedge \bigvee_{s' \in P_{ost}(s)\setminus\{s\}} \rho_{\Diamond\Box}(s'))$, where $\rho_{\Diamond\Box}(s)$ and $\rho_\bigcirc(s)$ are defined below.

Function $\rho_{\Diamond\Box}(\mathbf{s}, \mathbf{p})$*:* This function is used to identify the action to perform in order to avoid deadlocks and livelocks without violating φ. The agent can move to s only if $\rho_{\Diamond\Box}(s)$ holds. $\rho_{\Diamond\Box}(s) = \bigwedge_i \rho_{\Diamond\Box}(s, p_{2i}) \wedge \bigwedge_i \rho_{\Diamond\Box}(s, p_{3i}) \wedge \bigwedge_i \rho_{\Diamond\Box}(s, p_{4i}) \wedge \bigwedge_i \rho_{\Diamond\Box}(s, p_{5i})$, where $\rho_{\Diamond\Box}(s, p)$ is:

$$\rho_{\Diamond\Box}(s, p) = \begin{cases} true, & \text{if } V(s, p) = 0 \\ false, & \text{if } V(s, p) = \infty \\ \bigvee\limits_{s' \in P_{ost}(s)\setminus\{s\}} \Diamond\,\Box\,(AT_s \Rightarrow R(s')) \wedge \rho_{\Diamond\Box}(s') & \text{otherwise} \end{cases} \tag{11}$$

Intuitively, the agent from s can satisfy p only if the environment eventually allows the agent to move to a state

$s' \in P_{ost}(s) \setminus \{s\}$ such that it can satisfy the requirement function $R(s')$ and then move on to satisfy p.

Function $\rho_{\bigcirc}(\mathbf{s}, \mathbf{p})$*:* This function is used to identify the action to perform in order to avoid deadlocks and livelocks without violating φ. The agent can move to s only if $\rho_{\bigcirc}(s)$ holds.

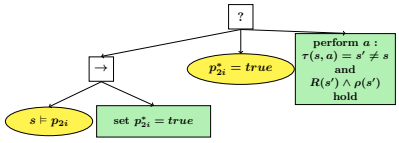
$\rho_{\bigcirc}(s) = \bigwedge_i \rho_{\bigcirc}(s, p_{2i}) \wedge \bigwedge_i \rho_{\bigcirc}(s, p_{3i}) \wedge \bigwedge_i \rho_{\bigcirc}(s, p_{4i}) \wedge \bigwedge_i \rho_{\bigcirc}(s, p_{5i})$, where $\rho_{\bigcirc}(s, p)$ is:

$$\rho_{\bigcirc}(s,p) = \begin{cases} (\Box(AT_s \Rightarrow \bigcirc R(s)) \vee \\ \vee \bigvee_{s' \in P_{ost}(s) \setminus \{s\}} \Box(AT_s \Rightarrow \bigcirc R(s'))), & \text{if } V(s,p) = 0 \\ false, & \text{if } V(s,p) = \infty \\ (\Box(AT_s \Rightarrow \bigcirc R(s)) \vee \bigvee\limits_{s' \in P_{ost}(s) \setminus \{s\}} \Box(AT_s \Rightarrow \bigcirc R(s'))) \wedge \\ \wedge \rho_{\bigcirc}(s') & \text{otherwise} \end{cases} \tag{12}$$

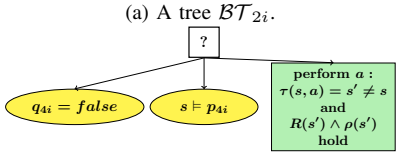
Intuitively, the agent can move to s only if at the next state it is allowed to stay in s or move to a state $s' \in P_{ost}(s) \setminus \{s\}$ such that it can satisfy the requirement function $R(s')$ and then move on to satisfy p.

Note that Assumption 2 can be formulated as: $\psi \Rightarrow R(s_0) \wedge \rho(s_0)$ while Assumption 3 can be formulated as: $\psi \Rightarrow \bigwedge_{\bar{s} \notin \Sigma(s_0)} R(\bar{s}) \wedge \rho_{\bigcirc}(\bar{s})$ and Assumption 4 can be formulated as: $\psi \Rightarrow \bigwedge_{\bar{s} \notin \Sigma(s_0)} R(\bar{s}) \wedge \rho_{\Diamond\Box}(\bar{s})$.

Synthesis of $\mathcal{BT}_2$*:* This tree executes the satisfying run for $\varphi_{\Diamond}$. Each proposition p_{2i} has to be satisfied at least once in the satisfying run. For each proposition we derive a BT $\mathcal{BT}_{2i}$ as in Figure 1(a). The tree $\mathcal{BT}_{2i}$, until p_{2i} is not satisfied, performs an action at each state s. The action leads to a state s' such that $R(s') \wedge \rho(s')$ holds.



(a) A tree $\mathcal{BT}_{2i}$.



(b) A tree $\mathcal{BT}_{4i}$

Fig. 1. Trees $\mathcal{BT}_{2i}$ and $\mathcal{BT}_{4i}$

Remark 3: It is possible that for a finite time there is no action a (see $\mathcal{BT}_{2i}$ in Figure 1(a)). In this case the tree does not perform any action (i.e. it performs the null action ε). Assumption 4 ensures that if the agent chose to move to s, then eventually an action a exists.

Synthesis of $\mathcal{BT}_3$*:* This tree executes the satisfying run for $\varphi_{\Rightarrow\Diamond}$. Each proposition p_{3i} has to be satisfied at least once in the satisfying run whenever q_{3i} holds. For each proposition we derive a BT $\mathcal{BT}_{3i}$ as in Figure 2. The tree $\mathcal{BT}_{3i}$, while q_{3i} is satisfied and until p_{3i} is not satisfied, performs an action at each state s. The action leads to a state s' such that $R(s') \wedge \rho(s')$ holds.

Synthesis of $\mathcal{BT}_4$*:* This tree executes the satisfying run for $\varphi_{\Rightarrow\Box\Diamond}$. Each propositions p_{4i} must hold infinitely often in the satisfying run whenever q_{4i} holds. For each proposition

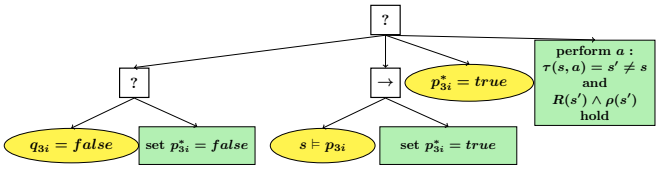
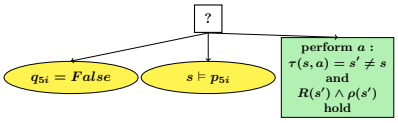


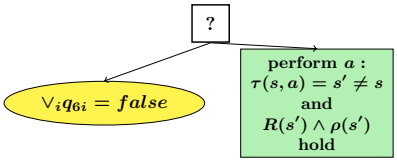
Fig. 2. A tree $\mathcal{BT}_{3i}$.

we derive a BT $\mathcal{BT}_{4i}$ as in Figure 1(b). The tree $\mathcal{BT}_{4i}$, whenever q_{4i} holds and until p_{4i} is not satisfied, performs an action at each state s. The action leads to a state s' such that $R(s')$ holds.

All the BTs $\mathcal{BT}_{4i}$ are composed in a sequence node with memory as the sub-trees $\mathcal{BT}_{4i}$ have to be executed in turn. They are composed in a sequence node with memory as the agent must be able to alternate runs that satisfy different p_{4i}.



(a) A tree $\mathcal{BT}_5$



(b) A tree $\mathcal{BT}_{6i}$

Fig. 3. Trees $\mathcal{BT}_5$ and $\mathcal{BT}_{6i}$

Synthesis of $\mathcal{BT}_5$*:* A proposition p_{5i} must eventually always hold whenever q_{5i} holds. We derive a BT $\mathcal{BT}_{5i}$ as in Figure 3(a).

The tree $\mathcal{BT}_{5i}$, whenever q_{5i} holds and until p_{5i} is not satisfied, performs an action at each state s. The action leads to a state s' such that $R(s')$ holds.

Synthesis of $\mathcal{BT}_6$*:* A proposition p_{6i} must hold at the next state whenever q_{6i} holds.

Multiple Possible Actions: If at state s there exist multiple actions a_i such that $a_i : \tau(s'_i, a_i) = s' \wedge s \neq s' \wedge R(s'_i) \wedge \rho(s'_i)$ holds, these action are collected in a fallback composition in ascending order of the value function at s'_i.

The final BT is

$$\mathcal{BT} = \text{sequence}(\mathcal{BT}_2, \mathcal{BT}_3, \mathcal{BT}_4, \mathcal{BT}_5, \mathcal{BT}_6). \tag{13}$$

Here we give an informal description of the execution of the $\mathcal{BT}$ in Equation (13). The root of $\mathcal{BT}$ generates the ticks. The tick first reaches the subtree $\mathcal{BT}_2$ (if any). $\mathcal{BT}_2$ is a sequence composition of $\mathcal{BT}_{2i}$. The tick reaches each $\mathcal{BT}_{2i}$, for each $\mathcal{BT}_{2i}$, the tick reaches the condition node $s \vDash p_{2i}$ (see Fig 1(a)). If that condition node returns success, the proposition p_{2i} is satisfied, the auxiliary variable p^*_{2i} is set to *true* and the success status is propagated back to $\mathcal{BT}$. If the condition $s \vDash p_{2i}$ returns failure then the tick reaches the condition node $p^*_{2i} = true$. If that condition returns success, the proposition p_{2i} was previously satisfied and the success

status is propagated back to $\mathcal{BT}$. If the condition node $p^*_{2i} = true$ returns failure the tick reaches the action node *perform* $a : \tau(s,a) = s' \neq s$ *and* $R(s') \wedge \rho(s')$ *hold* performing such action. The other subtrees are executed similarly.

VI. Theoretical Analysis

Proposition 1: Under Assumptions 1-4, a BT in form of (13) describes a policy that satisfies $\psi \Rightarrow \varphi_{\Box}$.

Proof: $\mathcal{BT}$ moves the agent to s' only if $R_{\Box}(s')$ holds. $R_{\Box}(s')$ holds if and only if $s' \vDash \varphi_{\Box}$. Hence $\mathcal{BT}$ will never move the agent to a state that violates $\varphi_{\Box}$. ■

Proposition 2: Under Assumptions 1-4, a BT in form of (13) describes a policy that satisfies $\psi \Rightarrow \varphi_{\Rightarrow\bigcirc}$.

Proof: At state s, $\mathcal{BT}$ executes an action a such that $s' = \tau(s,a)$ and $R(s') \wedge \rho(s')$ hold. When the agent is at state s for each p_{6i} in $\Box(q_{6i} \Rightarrow \bigcirc p_{6i})$ two cases occur: $\neg q_{6i}$ holds, or q_{6i} holds. In the former case, the specification $\Box(q_{6i} \Rightarrow \bigcirc p_{6i})$ is satisfied by definition; in the latter case $s' \vDash p_{6i}$ must hold.

Since the agent moved to s, $\rho_{\bigcirc}(s)$ holds. This ensures that at s for each i either $\neg q_{6i}$ or $s' \vDash p_{6i}$ holds.

$\psi \Rightarrow R(s_0) \wedge \rho(s_0)$ holds by Assumption 2. This ensures that there exists a satisfying run from s_0. $\psi \Rightarrow \bigwedge_{\bar{s} \notin \Sigma(s_0)} R(\bar{s}) \wedge \rho(\bar{s})$ holds by Assumptions 3 and 4. This ensures that if the environment forces the agent to move to $\bar{s}$, there exists a satisfying run for $\psi \Rightarrow \varphi_{\Rightarrow\bigcirc}$ from $\bar{s}$. ■

Proposition 3: Under Assumptions 1-4, a BT in form of (13) describes a policy that satisfies $\psi \Rightarrow \varphi_{\Diamond}$.

Proof: $\mathcal{BT}$ contains a number $|\varphi_{\Diamond}|$ of sub-trees $\mathcal{BT}_{2i}$. In each $\mathcal{BT}_{2i}$, if the proposition p_{2i} did not hold before (i.e. the condition node p^*_{2i} returns failure), it performs an action a. Two situations can occur: there exists a state $s' \in P_{ost}(s) \setminus \{s\}$ such that $R(s') \wedge \rho(s')$ holds, in this case a is such that $s' = \tau(s,a)$; or there is no such state, in this case a is the null action (the agent stays in s). We need to prove that eventually there exists a state $s' \in P_{ost}(s) \setminus \{s\}$ such that $R(s') \wedge \rho(s')$ holds. Since the agent moved to s, $\rho_{\Diamond\Box}(s, p_{2i})$ holds. This ensures that $s' \in P_{ost}(s) \setminus \{s\}$ such that $R(s') \wedge \rho(s')$ eventually holds. $\psi \Rightarrow R(s_0) \wedge \rho(s_0)$ holds by Assumptions 3 and 4. This ensures that there exists a satisfying run from s_0. $\psi \Rightarrow \bigwedge_{\bar{s} \notin \Sigma(s_0)} R(\bar{s}) \wedge \rho(\bar{s})$ holds by Assumption 4. This ensures that if the environment forces the agent to move to $\bar{s}$ there exists a satisfying run for φ from $\bar{s}$. Thus eventually, under Assumptions 1-4, there exists a state $s' \in P_{ost}(s) \setminus \{s\}$ such that $R(s') \wedge \rho(s')$ holds forever and from s' eventually we can always reach a state $\tilde{s}$ such that $\tilde{s} \vDash p_{2i}$. ■

Note that $R(s')$ is satisfied if, each $\Diamond p_{2i}$ was either satisfied in the past or a state $\tilde{s} \vDash p_{2i}$ is reachable from s' ($V(s', p_{2i}) \in \mathbb{N}$), hence $\mathcal{BT}$ executes a run that satisfies all the $\Diamond p_{2i}$. Moreover the satisfaction of the propositions $\varphi_{\Rightarrow\Box\Diamond}$ and $\varphi_{\Rightarrow\Diamond\Box}$ do not violate $\varphi_{\Diamond}$ as the BT to satisfy them is executed only after $\bigwedge_i p^*_{2i}$ holds.

Proposition 4: Under Assumptions 1-4, a BT in form of (13) describes a policy that satisfies $\psi \Rightarrow \varphi_{\Rightarrow\Box\Diamond}$.

Proof: $\mathcal{BT}$ contains a number $|\varphi_{\Rightarrow\Box\Diamond}|$ of sub-trees $\mathcal{BT}_{4i}$. Each $\mathcal{BT}_{4i}$ is derived to satisfy $\Box(q_{4i} \Rightarrow \Box\Diamond p_{4i})$. If the proposition q_{4i} holds and p_{4i} does not hold, it performs an action a. Two situations can occur: there exists a state $s' \in P_{ost}(s) \setminus \{s\}$ such that $R(s') \wedge \rho(s')$ holds, in this case a is such that $s' = \tau(s,a)$; or there is no such state, in this case a is the null action (the agent stays in s). We need to prove that eventually there exists a state $s' \in P_{ost}(s) \setminus \{s\}$ such that $R(s') \wedge \rho(s')$ holds. Since the agent moved to s, $\rho_{\Diamond\Box}(s, p_{4i})$ holds. This ensures that $s' \in P_{ost}(s) \setminus \{s\}$ such that $R(s') \wedge \rho(s')$ eventually holds. Moreover $\psi \Rightarrow R(s_0) \wedge \rho(s_0, p_{4i})$ holds by Assumption 2. This ensures that there exists a satisfying run from s_0. $\psi \Rightarrow \bigwedge_{\bar{s} \notin \Sigma(s_0)} R(\bar{s}) \wedge \rho(\bar{s})$ holds by Assumptions 3 and 4. This ensures that if the environment forces the agent to move to $\bar{s}$ there exists a satisfying run for φ from $\bar{s}$. Thus eventually , under Assumptions 1-4, there exists a state $s' \in P_{ost}(s) \setminus \{s\}$ such that $R(s') \wedge \rho(s')$ holds forever and from s' eventually we can always reach a state $\tilde{s}$ such that $\tilde{s} \vDash p_{4i}$.

We now need to prove that the satisfaction of the others $\Box(q_{4j} \Rightarrow \Box\Diamond p_{4j})$ in $\varphi_{\Rightarrow\Box\Diamond}$ do not violate the satisfaction of $\Box(q_{4i} \Rightarrow \Box\Diamond p_{4i})$. The agent is at s hence $R_{\Box\Diamond}(s)$ holds. $R_{\Box\Diamond}(s)$ holds only if $\Box(\neg q_{4i}) \vee \bigwedge_j (\exists \tilde{s} : V(\tilde{s}, p_{4j}) = 0 \wedge V(\tilde{s}, p_{4i}) \in \mathbb{N})$ holds. Hence either $\Box(q_{4i} \Rightarrow \Box\Diamond p_{4i})$ is trivially satisfied (i.e. $\Box(\neg q_{4i})$ holds) or the satisfaction of the other propositions $\Box(q_{4j} \Rightarrow \Box\Diamond p_{4j})$ do not violate $\Box(q_{4i} \Rightarrow \Box\Diamond p_{4i})$.

We now need to prove that the satisfaction of $\varphi_{\Diamond}$ do not violate $\Box(q_{4i} \Rightarrow \Box\Diamond p_{4i})$. $\mathcal{BT}_2$ is constructed to satisfy $\varphi_{\Diamond}$. $\mathcal{BT}_2$ let the agent move to a state s only if $R_{\Box\Diamond}(s)$ holds.

We now need to prove that the satisfaction of $\varphi_{\Rightarrow\Diamond}$ do not violate $\Box(q_{4i} \Rightarrow \Box\Diamond p_{4i})$. $\mathcal{BT}_3$ is constructed to satisfy $\varphi_{\Diamond}$. $\mathcal{BT}_3$ let the agent move to a state s only if $R_{\Box\Diamond}(s)$ holds.

We now need to prove the satisfaction of $\varphi_{\Rightarrow\Diamond\Box}$ do not violate $\Box(q_{4i} \Rightarrow \Box\Diamond p_{4i})$. The agent is at s hence $R_{\Diamond\Box}(s)$ holds. $R_{\Diamond\Box}(s)$ holds only if for each k in $\Box(q_{5k} \Rightarrow \Diamond \Box\, p_{5k})$ the following holds: $\Box\Diamond\neg q_{5k} \vee \Diamond\Box\neg q_{4i} \vee \bigwedge_j \exists \tilde{s} : V_{p_{5i}}(\tilde{s}, p_{4j}) = 0 \wedge V_{p_{5i}}(\tilde{s}, p_{4i}) \in \mathbb{N}$, hence the agent can either violate p_{5k} infinitely often (i.e. $\Box\Diamond\neg q_{5k}$ holds) or the agent can violate p_{4i} eventually forever (i.e. $\Diamond\,\Box\,\neg q_{4i}$ holds) or the agent can satisfy $\psi \Rightarrow \varphi_{\Rightarrow\Box\Diamond}$ and p_{5i} (i.e. $\bigwedge_j \exists \tilde{s} : V_{p_{5i}}(\tilde{s}, p_{4j}) = 0 \wedge V_{p_{5i}}(\tilde{s}, p_{4i}) \in \mathbb{N}$ holds.) ■

Proposition 5: Under Assumptions 1-4, a BT in form of (13) describes a policy that satisfies $\psi \Rightarrow \varphi_{\Rightarrow\Diamond}$.

Proof: $\mathcal{BT}$ contains a number $|\varphi_{\Rightarrow\Diamond}|$ of sub-trees $\mathcal{BT}_{3i}$. Each $\mathcal{BT}_{3i}$ is derived to satisfies $\Box(q_{3i} \Rightarrow \Diamond p_{3i})$. If the proposition q_{3i} holds and p_{3i} does not hold, it performs an action a. Two situations can occur: there exists a state $s' \in P_{ost}(s) \setminus \{s\}$ such that $R(s') \wedge \rho(s')$ holds, in this case a is such that $s' = \tau(s,a)$; or there is no such state holds, in this case a is the null action (the agent stays in s). We need to prove that there exists a state $s' \in P_{ost}(s) \setminus \{s\}$ such that $R(s') \wedge \rho(s')$ holds infinitely often. Since the agent moved to s, $\rho_{\Diamond\Box}(s)$ holds. This ensures that $s' \in P_{ost}(s) \setminus \{s\}$ such that $R(s') \wedge \rho(s')$ eventually holds.

$\psi \Rightarrow R(s_0) \wedge \rho(s_0, p_{3i})$ holds by Assumption 2. This ensures that there exists a satisfying run from s_0. $\psi \Rightarrow \bigwedge_{\bar{s} \notin \Sigma(s_0)} \rho(\bar{s})$ holds by Assumptions 3 and 4. This ensures that if the environment forces the agent to move to $\bar{s}$ there

exists a satisfying run for φ from $\bar{s}$. Thus eventually, under Assumptions 1-4, there exists a state $s' \in P_{ost}(s) \setminus \{s\}$ such that $R(s') \wedge \rho(s')$ holds forever and from s' eventually we can always reach a state $\tilde{s}$ such that $\tilde{s} \vDash p_{3i}$.

We now need to prove that the satisfaction of the others $\square(q_{3j} \Rightarrow \lozenge p_{3j})$ in $\varphi_{\Rightarrow\lozenge}$ do not violate the satisfaction of $\square(q_{4i} \Rightarrow \lozenge p_{4i})$. The agent is at s hence $R_{\Rightarrow\lozenge}(s)$ holds. $R_{\Rightarrow\lozenge}(s)$ holds only if $\square(\neg q_{3i}) \vee p^*_{3i} \vee \bigwedge_j (\exists \tilde{s} : V(\tilde{s}, p_{3j}) = 0 \wedge V(\tilde{s}, p_{3i}) \in \mathbb{N})$ holds. Hence either $\square(q_{3i} \Rightarrow \square\lozenge p_{3i})$ is trivially satisfied (i.e. $\square(\neg q_{3i})$ holds) or the satisfaction of the other propositions $\square(q_{3j} \Rightarrow \square\lozenge p_{3j})$ do not violate $\square(q_{3i} \Rightarrow \square\lozenge p_{3i})$ or p_{3i} was satisfied since q_{3i} started to hold (i.e. p^*_{3i} hold).

We now need to prove that the satisfaction of $\varphi_\lozenge$ do not violate $\square(q_{3i} \Rightarrow \lozenge p_{3i})$. $\mathcal{BT}_2$ is constructed to satisfy $\varphi_\lozenge$. $\mathcal{BT}_2$ let the agent move to a state s only if $R_{\Rightarrow\lozenge}(s)$ holds.

We now need to prove that the satisfaction of $\varphi_{\square\lozenge}$ do not violate $\square(q_{4i} \Rightarrow \lozenge p_{4i})$. $\mathcal{BT}_4$ is constructed to satisfy $\varphi_{\square\lozenge}$. $\mathcal{BT}_4$ let the agent move to a state s only if $R_{\Rightarrow\lozenge}(s)$ holds.

We now need to prove that the satisfaction of $\varphi_{\Rightarrow\lozenge\square}$ do not violate $\square(q_{3i} \Rightarrow \lozenge p_{3i})$. The agent is at s hence $R_{\lozenge\square}(s)$ holds. $R_{\lozenge\square}(s)$ holds only if for each k in $\square(q_{5k} \Rightarrow \lozenge \square p_{5k})$ the following holds: $\square\lozenge\neg q_{5k} \vee \lozenge\square\neg q_{3i} \vee \bigwedge_j \exists \tilde{s} : V_{p_{5i}}(\tilde{s}, p_{3j}) = 0 \wedge V_{p_{3i}}(\tilde{s}, p_{3i}) \in \mathbb{N}$, hence the agent can either violate p_{5k} infinitely often (i.e. $\square\lozenge\neg q_{5k}$ holds) or the agent can violate p_{3i} eventually forever (i.e. $\lozenge \square \neg q_{4i}$ holds) or the agent can satisfy $\psi \Rightarrow \varphi_{\Rightarrow\lozenge}$ and p_{5i} (i.e. $\bigwedge_j \exists \tilde{s} : V_{p_{5i}}(\tilde{s}, p_{3j}) = 0 \wedge V_{p_{5i}}(\tilde{s}, p_{3i}) \in \mathbb{N}$ holds.) ■

Proposition 6: Under Assumptions 1-4, a BT in form of (13) describes a policy that satisfies $\psi \Rightarrow \varphi_{\Rightarrow\lozenge\square}$

Proof: $\mathcal{BT}$ contains a number $|\varphi_{\Rightarrow\lozenge\square}|$ of sub-trees $\mathcal{BT}_{5i}$. Each $\mathcal{BT}_{5i}$, if the proposition q_{5i} holds and p_{5i} does not hold, performs an action a. Two situations can occur: there exists a state $s' \in P_{ost}(s) \setminus \{s\}$ such that $R(s') \wedge \rho(s')$ holds, in this case a is such that $s' = \tau(s, a)$; or there is no such state, in this case a is the null action (the agent stays in s). We need to prove that there exists a state $s' \in P_{ost}(s) \setminus \{s\}$ such that $R(s') \wedge \rho(s')$ eventually holds. Since the agent moved to s, $\rho_{\lozenge\square}(s, p_{5i})$ holds. This ensures that $s' \in P_{ost}(s) \setminus \{s\}$ such that $R(s') \wedge \rho(s')$ eventually holds.

$\psi \Rightarrow R(s_0) \wedge \rho(s_0, p_{5i})$ holds by Assumption 2 . This ensures that there exists a satisfying run from s_0. $\psi \Rightarrow \bigwedge_{\bar{s} \notin \Sigma(s_0)} R(\bar{s}) \wedge \rho(\bar{s})$ holds by Assumptions 3 and 4. This ensures that if the environment forces the agent to move to $\bar{s}$ (from which every satisfying run includes s_0) there exists a satisfying run for φ from $\bar{s}$ to s_0. Thus eventually there exists a state $s' \in P_{ost}(s) \setminus \{s\}$ such that $R(s') \wedge \rho(s')$ holds forever and from s' eventually we can always reach a state $\tilde{s}$ such that $\tilde{s} \vDash p_{5i}$.

We now need to prove that the satisfaction of $\varphi_\lozenge$ does not violate $\square(q_{5i} \Rightarrow \lozenge \square\, p_{5i})$. $\mathcal{BT}_2$ is constructed to satisfy $\varphi_\lozenge$. $\mathcal{BT}_2$ let the agent move to a state s only if $R_{\lozenge\square}(s)$ holds.

We now need to prove that the satisfaction of the other $\square(q_{5j} \Rightarrow \lozenge \square\, p_{5j})$ in $\varphi_{\Rightarrow\lozenge\square}$ does not violate the satisfaction of $\square(q_{5i} \Rightarrow \lozenge\ \square p_{5i})$. The agent is at s hence $R_{\lozenge\square}(s)$ holds. $R_{\lozenge\square}(s)$ holds only if there exists a state $\tilde{s}$ reachable from s such that $\tilde{s} \vDash p_{5i}$ and $\lozenge \square \neg q_{5i} \vee \bigwedge_{j:s\not\vDash} \lozenge \square \neg q_{5i}$ hold. That is eventually forever the agent will not reach any state that does not satisfy $\varphi_{\Rightarrow\lozenge\square}$.

We now need to prove that the satisfaction of $\varphi_{\Rightarrow\lozenge}$ does not violate $\square(q_{5i} \Rightarrow \lozenge\square p_{5i})$. $R_{\lozenge\square}(s)$ holds only if $(\square\lozenge\neg q_{5i} \vee \bigwedge_j (\lozenge \square \neg q_{3j} \vee \bigwedge_{k\neq j} \lozenge \square \neg q_{3k} \vee (\exists s' : V_{p_{5i}}(s', p_{3k}) = 0 \wedge V_{p_{5i}}(s', p_{3j}) \in \mathbb{N})))$ holds. That is the agent is either allowed to leave $\mathcal{T}_{p_{5i}}$ infinitely often to satisfy $\varphi_{\square\lozenge}$ (i.e. $\square\lozenge\neg q_{5i}$ holds) or it can satisfy $\varphi_{\Rightarrow\lozenge}$ without leaving $\mathcal{T}_{p_{5i}}$.

We now need to prove the satisfaction of $\varphi_{\square\lozenge}$ does not violate $\square(q_{5i} \Rightarrow \lozenge \square\, p_{5i})$. $R_{\lozenge\square}(s)$ holds only if $(\square\lozenge\neg q_{5i} \vee \bigwedge_j (\lozenge \square \neg q_{4j} \vee \bigwedge_{k\neq j} \lozenge \square \neg q_{4k} \vee (\exists s' : V_{p_{5i}}(s', p_{4k}) = 0 \wedge V_{p_{5i}}(s', p_{4j}) \in \mathbb{N})))$ holds. That is the agent is either allowed to leave $\mathcal{T}_{p_{5i}}$ infinitely often to satisfy $\varphi_{\square\lozenge}$ (i.e. $\square\lozenge\neg q_{5i}$ holds) or it can satisfy $\varphi_{\square\lozenge}$ without leaving $\mathcal{T}_{p_{5i}}$. ■

Theorem 1: The policy π described by $\mathcal{BT}$ in form of (13) solves Problem 1. *Proof:*

A BT that satisfies all of following specifications: $\psi \Rightarrow \varphi_\square$, $\psi \Rightarrow \varphi_\lozenge$, $\psi \Rightarrow \varphi_{\Rightarrow\lozenge}$, $\psi \Rightarrow \varphi_{\Rightarrow\lozenge\square}$, $\psi \Rightarrow \varphi_{\Rightarrow\square\lozenge}$, $\psi \Rightarrow \varphi_{\Rightarrow\bigcirc}$ solves Problem 1. From Propositions 1 to 6, $\mathcal{BT}$ satisfies all the aforementioned specifications. ■

Proposition 7: The $\mathcal{BT}$ for φ is computed in $O(|\varphi||\mathcal{T}| + |\varphi|^3 + |\varphi||\mathcal{T}|log(|\mathcal{T}|))$ time.

Proof: Let $|\mathcal{T}|$ be the number of state and edges of $\mathcal{T}$, the value functions are computed in $O(|\mathcal{T}|)$ time. The requirement function $R_\square$ is computed in $O(|\varphi_\square|)$ time. The requirement function $R_\lozenge$ is computed in $O(|\varphi_\lozenge|)$ time. Similarly $R_\bigcirc$ is computed in $O(|\varphi_{\Rightarrow\bigcirc}|)$ time, $R_{\Rightarrow\square\lozenge}$ is computed in $O(|\varphi_{\Rightarrow\square\lozenge}||\varphi_{\Rightarrow\lozenge\square}|(|\varphi_{\Rightarrow\square\lozenge}| + |\varphi_{\Rightarrow\lozenge\square}|))$, and $R_{\Rightarrow\lozenge\square}$ is computed in $O(|\varphi_{\Rightarrow\square\lozenge}||\varphi_{\Rightarrow\lozenge\square}||\varphi_{\Rightarrow\square\lozenge}| + |\varphi_{\Rightarrow\lozenge\square}|))$. $\rho_\bigcirc$ and ρ are both computed in $O(|\varphi_{\Rightarrow\bigcirc}|(|\varphi_\lozenge| + |\varphi_{\Rightarrow\square\lozenge}| + |\varphi_{\Rightarrow\lozenge\square}|))$ time. The satisfaction set $\Sigma(s_0)$ is computed in $O(|\varphi||\mathcal{T}|)$ time. For each sub-tree $\mathcal{BT}_i$ the actions are sorted by the value function. The sorting is done in $O(|\mathcal{T}|log(|\mathcal{T}|))$ for each formula in φ. Hence the $\mathcal{BT}$ for φ is computed in in $O(|\varphi||\mathcal{T}| + |\varphi|^3 + |\varphi||\mathcal{T}|log(|\mathcal{T}|))$ time. ■

VII. Results

In this section we illustrate the proposed framework for different tasks.

Consider a gridworld scenario with 15×26 cells, similar to the one used in [16]. A robot occupies a single cell at a time and it can move to one of the four adjacent cells (in the four cardinal directions) and the environment operates some atomic propositions in a non-deterministic fashion. A video showing the execution of each task is publicly available.[1]

Example 1: Consider the scenario in Figure 4(a), where the gray cells are obstacles/walls and each room satisfies one of the following atomic propositions: A, B, C, D, E, F, G, or H as labeled in the figure. The task is to reach room A and B then repeatedly visit C, E, and F, whenever the $alarm$ is turned on, the system must not enter neither B nor C. $alarm$ is an atomic proposition driven by the environment. The given task can be described by the following LTL formula: $\varphi = \square\neg obstacle \wedge \lozenge A \wedge \lozenge B \wedge \square\lozenge E \wedge \square\lozenge F \wedge \square\lozenge C \wedge \square(alarm \Rightarrow$

[1] https://youtu.be/RCuQopEQD3M

$\neg B) \wedge \square(alarm \Rightarrow \neg C)$. The computed path is shown in Figure 4(a). We refer to the video available to visualize the execution.

Example 2: Consider the scenario in Figure 4(b), where the gray cells are obstacles, the white hashed cells are one way cells (it can be reached only from the cell below it) and the atomic propositions are: p_1; p_2; p_3; and p_4. The areas and the rooms are labeled with the atomic propositions satisfied.

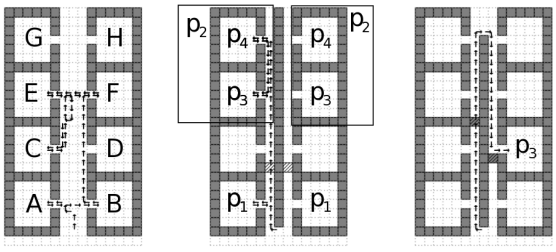
The task is defined by the following LTL formula: $\varphi = \square\neg obstacle \wedge \lozenge p_1 \wedge \lozenge \square\, p_2 \wedge \square\lozenge p_3 \wedge \square\lozenge p_4$.

Running our framework we obtain the following environmental constraint (Assumption 2-4): $\psi \Rightarrow true$ that is trivially satisfied. This is because the system behavior is not influenced by the environmental variables. The computed path is shown in Figure 4(b), note that the only choice the systems has is to go to the left hand side, since the corridor on the right hand side do not satisfy p_2. We refer to the video available to visualize the execution.

Example 3: Consider the scenario in Figure 4(c), where the gray cells are obstacles/walls, the hashed gray cells are traps (being either active or inactive) and the atomic propositions are: $trap_1$, $trap_2$, and p_3 while the atomic propositions for the environment are $warning_1$ and $warning_2$. $warning_i$ hold if the $trap_i$ will hold at the next state.

The task is defined by the following LTL formula: $\varphi = \square\neg obstacle \wedge \lozenge p_3 \wedge \square(warning_1 \Rightarrow \bigcirc\neg trap_1) \wedge \square(warning_2 \Rightarrow \bigcirc\neg trap_2)$.

Running our framework we obtain the following environmental constraint (Assumption 2-4): $\psi \Rightarrow \lozenge\, \square\, (AT_{sb1} \Rightarrow \bigcirc\neg warning_1) \vee \lozenge\, \square\, (AT_{sb2} \Rightarrow \bigcirc\neg warning_2)$ (where sbi is the cell below the trap i) that eventually, either $trap_1$ or $trap_2$ must be inactive when the agent is in the cell below the cell with the trap. The environment satisfies the following $\psi \Rightarrow \lozenge\, \square\, (AT_{sb1} \Rightarrow \bigcirc\neg trap_1)$ hence the system chooses the path on the left hand side and it waits in front of the obstacle until this disappears. The computed path is shown in Figure 4(c), note that the only choice the systems has is to go to the left hand side, $trap_2$ is not guaranteed to become inactive.



(a) Path computed for Example 1. (b) Path computed for Example 2. (c) Path computed for Example 3

Fig. 4. Path computed for each motion planning scenario.

VIII. Conclusions

We proposed the synthesis of a correct-by-construction BT in polynomial time. We showed how using a task specification language is it possible to derive a BT that *guarantees* that a task is completed. We analyzed the framework from a theoretical standpoint and showed some examples.

IX. Acknowledgments

This work has been supported by the SARAFun project, partially funded by the EU within H2020-ICT-2014. We thank Ioannis Filippidis for his valuable input into this paper.

References

[1] I. Millington and J. Funge, *Artificial intelligence for games*. CRC Press, 2009.

[2] S. Rabin, *Game AI Pro*. CRC Press, 2014, ch. 6. The Behavior Tree Starter Kit.

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

[4] K. R. Guerin, C. Lea, C. Paxton, and G. D. Hager, "A framework for end-user instruction of a robot assistant for manufacturing," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2015.

[5] D. Hu, Y. Gong, B. Hannaford, and E. J. Seibel, "Semi-autonomous simulated brain tumor ablation with raven ii surgical robot using behavior tree," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2015.

[6] A. J. Champandard, M. Dawe, and D. Hernandez-Cerpa, "Behavior trees: three ways of cultivating game ai," in *Game Developers Conference, AI Summit*, 2010.

[7] M. Colledanchise and P. Ögren, "How Behavior Trees Modularize Hybrid Control Systems and Generalize Sequential Behavior Compositions, the Subsumption Architecture and Decision Trees," *IEEE Transactions on Robotics*, vol. 33, no. 2, pp. 372–389, April 2017.

[8] M. Colledanchise and P. Ögren, "How Behavior Trees Generalize the Teleo-Reactive Paradigm and And-Or-Trees," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016.

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

[10] M. Nicolau, D. Perez-Liebana, M. O'Neill, and A. Brabazon, "Evolutionary behavior tree approaches for navigating platform games," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. PP, no. 99, pp. 1–1, 2016.

[11] A. Pnueli, "The temporal logic of programs," in *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*. Washington, DC, USA: IEEE Computer Society, 1977, pp. 46–57.

[12] T. Wongpiromsarn, U. Topcu, N. Ozay, H. Xu, and R. M. Murray, "Tulip: a software toolbox for receding horizon temporal logic planning," in *Proceedings of the 14th international conference on Hybrid systems: computation and control*. ACM, 2011, pp. 313–314.

[13] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "Temporal-logic-based reactive mission and motion planning," *IEEE transactions on robotics*, vol. 25, no. 6, pp. 1370–1381, 2009.

[14] N. Piterman, A. Pnueli, and Y. Saar, "Synthesis of reactive (1) designs," in *International Workshop on Verification, Model Checking, and Abstract Interpretation*. Springer, 2006, pp. 364–380.

[15] I. Filippidis and R. M. Murray, "Symbolic construction of GR(1) contracts for synchronous systems with full information," *CoRR*, vol. abs/1508.02705, 2015.

[16] E. M. Wolff, U. Topcu, and R. M. Murray, "Efficient reactive controller synthesis for a fragment of linear temporal logic," in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, May 2013, pp. 5033–5040.

[17] G. E. Fainekos, H. Kress-Gazit, and G. J. Pappas, "Hybrid controllers for path planning: A temporal logic approach," in *44th IEEE Conference on Decision and Control*. IEEE, 2005, pp. 4885–4890.

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

[19] R. Dey and C. Child, "Ql-bt: Enhancing behaviour tree design and implementation with q-learning," in *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*. IEEE, 2013, pp. 1–8.

[20] M. Colledanchise, R. Parasuraman, and P. Ögren, "Learning of behavior trees for autonomous agents," *arXiv preprint 1504.05811*, 2015.

[21] J. Tumova, A. Marzinotto, D. V. Dimarogonas, and D. Kragic, "Maximally satisfying ltl action planning," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2014, pp. 1503–1510.

[22] C. Baier and J.-P. Katoen, *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.