Conditional Behavior Trees: Definition, Executability, and Applications

Eleonora Giunchiglia, Michele Colledanchise, Lorenzo Natale and Armando Tacchella

***Abstract*— Behavior Trees (BTs) are gaining acceptance in robotics to specify action policies at the deliberative level. Their advantages include modularity, ease of use and increasing tool support. In this paper, we define Conditional Behavior Trees (CBTs) as an extension of BTs wherein actions are decorated considering pre- and post-conditions. CBTs improve on basic BTs in that they enable monitoring the execution of single actions by checking pre- and post-conditions, respectively. Since there might exist action sequences wherein some pre-conditions are violated, CBT executability may depend on the success/failure of specific actions. We developed an encoding of CBT executability into satisfiability of propositional formulas to be checked off-line in a publicly-available tool that computes the encoding for generic CBTs. For the kind of application scenarios and related behavior specifications that we consider, we show that our approach is effective and yields formal guarantees about the executability of deliberative policies designed as CBTs.**

I. Introduction

A Behavior Tree (BT) is a way to structure the switching between different actions in an autonomous agent. Actions are assumed to be re-usable sub-activities that the robot carries out to complete an overall task. BTs were developed in the computer game industry, as a tool to increase modularity in the control structures of non-player characters (NPCs) [1] to enable modeling of discrete reactive behavior. In the last decade, BTs have found appreciation also in the robotics community, including unmanned aerial vehicles [2], [3], [4], medical robotics [5], industrial robotics [6], and AI [7], [8]. Moreover, BTs generalize successful robot control architectures such as the Subsumption Architecture, Decision Trees [9] and the Teleo-reactive Paradigm [10]. Modularity in BTs is exploited using "prune and graft" operations, whereby actions or entire subtrees can be inserted, appended or removed from a BT without worrying to maintain consistency with other parts of the BT. In other words, individual behaviors can be reused in the context of another higher-level behavior, without needing to specify how they relate to subsequent behaviors [11]. Additional advantages of BTs include ease of use and increasing tool support — see [9] for a recent survey.

An example of a BT performing a simple activity is shown in Figure 1. Informally, inner nodes represent control flow statements and leaves represent action statements that can succeed or fail when executed by the agent. With reference to Figure 1, while actions have the obvious meaning, the control nodes labeled with "?", "$\rightarrow$" and "$\rightrightarrows$" define a fallback node, a sequence node, and a parallel node, respectively. Fallback nodes execute their children one after the other, and they are successful as long as at least one of their child is so; sequence nodes also execute their children one after the other, but they are successful only if all their children are successful; parallel nodes execute their children simultaneously and require all of them to be successful in order to complete successfully. Therefore, the BT in Figure 1 defines a policy whereby the agent tries to reach the kitchen, to locate the bottle, and to fetch it while tracking it. In case of failure of the main course of actions, the agent asks a human for help.

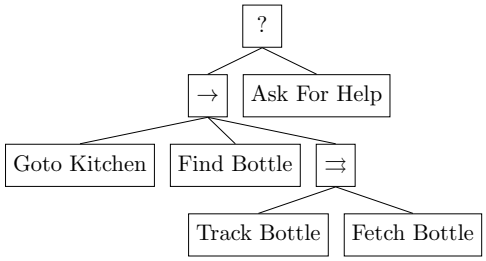


Fig. 1. A Behavior Tree (BT) coordinating the execution of five sub-behaviors.

When agents are deployed in the physical world, actions may fail because enabling conditions are not met. At the same time, once an action is executed successfully, the agent can usually assume that the state of the world changed in a predictable way. We call action *pre-conditions* the set of facts that ought to hold for an action to be executable, and action *post-conditions* the facts that hold after an action is executed. While classical AI planning formalisms, e.g., STRIPS [12] and its more recent evolution PDDL [13], cater for action pre- and post-conditions, BTs do not. Policies synthesized from scenarios described in STRIPS/PDDL can be subject to monitoring in order to ensure that pre-conditions are met before executing an action and that post-conditions are enforced after action execution — see, e.g., [14]. On the other hand, it is not possible to couple a monitor to a BT in order to supervise the correct execution of actions.

To fill the gap between specification and monitoring in BTs, we define Conditional Behavior Trees (CBTs). Intuitively, CBTs are an extension of BTs wherein actions are decorated considering pre- and post-conditions as in classical AI planning. CBTs improve on basic BTs since they

Eleonora Giunchiglia (eleonora.giunchiglia@cs.ox.ac.uk) is with University of Oxford, Department of Computer Science, Wolfson Building, Parks Road, Oxford OX13QD.

Armando Tacchella (armando.tacchella@unige.it) is with Università degli Studi di Genova, DIBRIS, Viale Causa 13, 16145 Genova

Michele Colledanchise (michele.colledanchise@iit.it) and Lorenzo Natale (lorenzo.natale@iit.it) are with Istituto Italiano di Tecnologia, via Morego 30, 16163 Genova.

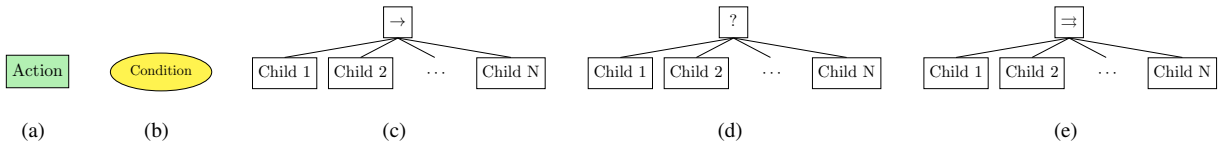


Fig. 2. Graphical representation of each type of BT nodes: 2(a) action node, 2(b) condition node, 2(c) sequence node with N children. 2(d) fallback node with N children, and 2(e) Parallel node with N children. A child can be any BT node.

enable off-line verification for executability and reduce on-line monitoring to check that single actions fulfill their pre- and post-conditions. CBTs retain all advantages exhibited by BTs, e.g., it is still possible to prune and graft actions or sub-trees without worrying about implicit behavioral interactions. However, if we view a CBT as a compact encoding of several action sequences, we observe that there might be sequences wherein some pre-conditions are always violated. Since we assume that post-conditions are enforced only when an action is successful, the result is that the executability of a sequence of actions in a CBT may depend on the success/failure of specific actions. Notice that the success or failure of the root node of a BT is independent of executability: a CBT which is verified to be executable off-line, can still return failure; on the other hand, a BT which is not executable could in principle return success by never attempting to execute the action whose preconditions are violated. Because of this, refactoring a CBT whose actions share pre- and post-conditions is to be checked for executability to avoid unpredictable behavior at runtime.

We present an off-line checking method to verify the executability of a CBT, i.e., ensure that all action sequences encoded by the CBT are executable. We wish to stress that our method does not aim to replace on-line monitoring of pre- and post-conditions for single actions, but it guarantees that if such monitoring does not signal anomalies, then the robot will operate correctly. The keystone of our approach is an efficient encoding of a CBT executability problem into a propositional satisfiability (SAT) problem. Such encoding is not supported by the standard BT formalism. Informally speaking, given any CBT, we can construct a Boolean formula that admits a satisfying assignment exactly when there is at least one sequence that is always *not* executable. From the satisfying assignments (if any) it is also possible to reconstruct the problematic action sequence to debug the CBT. The encoding is efficient because its size is at most polynomial in the size of the behavior tree. Furthermore, while SAT is the archetypal NP-hard problem [15], and thus the worst-case complexity of checking executability could be exponential in the size of the CBT, current state-of-the-art solvers — see, e.g., [16] — can handle formulas with variables in the order of 10^6 in a matter of seconds. In all the scenarios that we consider, and for the (C)BTs that are typically found in the literature, the resulting encodings are not challenging. Indeed, our tool is able to compute the encodings and check the executability of the CBTs proposed in the paper in less than 50 ms. Early study of formal verification of BTs are found in [3] where the BT is parsed into the formalism *Attributive Language with Complements and concrete Domains ALC(D)*, for tractability purposes. The verification is executed on a pre-existent BT. Other approaches combine formal verification with the BT modeling to create a *correct-by-construction* BT, where the framework automatically synthesizes a BT to satisfy a user-defined Linear Temporal Logic (LTL) formula. An early approach [17] synthesizes a *maximally-satisfying* control policy in form of a BT taking into account robot failures. However the BT modeling are used only as a bridge between their task execution framework and the low level controllers of the robot. Another approach [18] synthesizes the BT directly, preserving the advantages of BTs in terms of modularity, reactiveness, robustness and human readability. A recent extension on the BT formalism [19] allows the implementation of a PDDL-like planner that gives a BT as a result. In our work it is enough to define pre- and post-conditions to formulate the SAT problem.

Summing up, in this paper we contribute *(i)* a definition of CBTs, an extension of BTs that enables expressing pre- and post-conditions for actions, *(ii)* a methodology to statically check CBTs to ensure their executability, and *(iii)* a tool based on off-the-shelf SAT solvers, which is publicly available. [1] Finally, we contribute the description of some human-robot interaction scenarios devised in the EU project RobMoSys/CARVE[2] and based on the R1 robot [20]. With these scenarios, we show that checking executability is totally feasible for CBTs of practical use. The remainder of the paper is structured as follows. In Section II we provide syntax and semantics of BTs; in Section III we define the syntax and semantics of CBTs, including the notion of executability; in Section IV we show how to encode executability into SAT, and in Section V we test our encoding on the RobMoSys/CARVE scenarios. We conclude the paper in Section VI with some final remarks.

II. Behavior Trees

As explained in [9], BTs are rooted trees whose inner nodes are *control flow nodes* and whose leaves are *execution nodes*. BTs present two types of execution nodes, namely *action* and *condition*, and four types of control flow nodes, namely *sequence*, *fallback*, *parallel* and *decorator*. Since the decorator node carries user-defined behavior, we do not consider it in our analysis. For each type of node we consider, we show the corresponding graphical syntax in Figure 2.

[1] https://github.com/EGiunchiglia/CBTs-Checker.
[2] Website: https://robmosys.eu/carve/.

Algorithm 1 Pseudocode of the function Tick() of a Sequence Node

```
for i ← 1 to N do
    childStatus ← child(i).Tick()
    if childStatus = RUNNING then
        return RUNNING
    else if childStatus = FAILURE then
        return FAILURE
return SUCCESS
```

Algorithm 2 Pseudocode of the function Tick() of Fallback Node

```
for i ← 1 to N do
    childStatus ← child(i).Tick()
    if childStatus = RUNNING then
        return RUNNING
    else if childStatus = SUCCESS then
        return SUCCESS
return FAILURE
```

Algorithm 3 Pseudocode of the function Tick() of Parallel Node

```
for i ← 1 to N do
    childStatus ← child(i).Tick()
if ∀i.(childStatus(i) = SUCCESS) then
    return SUCCESS
else if ∃i.(childStatus(i) = FAILURE) then
    return FAILURE
return RUNNING
```

Fig. 3. Pseudocode for the semantics of control flow nodes in BTs.

The execution of every BT starts from the root, which is assumed to receive *ticks* with a given frequency. Ticks are then propagated from each parent node to its children, and each node is executed exactly when it receives ticks. Once the tick is received, each node may return: *(i)* RUNNING if its execution is not complete yet, *(ii)* SUCCESS if the execution was completed successfully, or *(iii)* FAILURE otherwise. The execution semantics of action nodes is straightforward: when such nodes receive ticks, they execute the corresponding action returning SUCCESS if the action is completed, FAILURE if the action fails, and RUNNING while the action is ongoing. When a condition node receives a tick, it checks if a condition holds or not: in the former case it returns SUCCESS, in the latter it returns FAILURE, but it never returns RUNNING. The execution semantics of control flow nodes is given in Figure 3, assuming that a node has N children.

III. Conditional Behavior Trees

CBTs extend BTs by specifying, for each action node, *(i)* a set of *pre-conditions* that must be satisfied in order to perform the corresponding action, and *(ii)* a set of *post-conditions* that are satisfied exactly when the action is successful. To define the syntax of CBTs, let us consider two finite and disjoint sets:

1) the set of actions $\mathcal{A}$ that a robot may perform, and
2) the set of fluents $\mathcal{F}$, i.e., propositions about the robot and the environment that might hold or not depending on the current state.

For each fluent $p \in \mathcal{F}$ we define its *complement* as $\overline{p}$, and the set $\overline{\mathcal{F}} = \{\overline{p} \mid p \in \mathcal{F}\}$ is the set of all complements of $\mathcal{F}$. The set of *literals* $\mathcal{C}$ is defined as $\mathcal{C} = \mathcal{F} \cup \overline{\mathcal{F}}$. Every action node $a \in \mathcal{A}$ is decorated by a couple $(Pre_a, Post_a)$, where, for all actions $a \in \mathcal{A}$, both Pre_a and $Post_a$ are subsets of $\mathcal{F}$ such that for all $p \in \mathcal{F}$ it is never the case that $p \in Pre_a$ (resp. $p \in Post_a$) when $\overline{p} \in Pre_a$ (resp. $\overline{p} \in Post_a$) and vice-versa.

Example 1: Consider the BT represented in Figure 1. The set of actions $\mathcal{A}$ is defined as

$$\mathcal{A} = \{GK, FB, TB, FeB, AH\}.$$

where the elements of $\mathcal{A}$ stand for "Goto Kitchen", "Find Bottle", "Track Bottle", "Fetch Bottle", and "Ask for Help", respectively. We define the set of fluents $\mathcal{F}$ as

$$\mathcal{F} = \{rk, bv, bl, bf, nh\}$$

where each proposition stands for the following facts:

- rk: the robot is in the kitchen;
- bv: the bottle is visible;
- bl: the bottle is located;
- bf: the bottle is fetched; and
- nh: the robot is near a human.

Finally, we may obtain a CBT by decorating actions as follows:

- $Pre_{GK} = \{nh\}$ and $Post_{GK} = \{rk, bv, \neg nh\}$;
- $Pre_{FB} = \{rk, bv\}$ and $Post_{FB} = \{bl\}$;
- $Pre_{TB} = \{rk, bl\}$ and $Post_{TB} = \{\}$;
- $Pre_{FeB} = \{rk, bl\}$ and $Post_{FeB} = \{bf\}$;
- $Pre_{AH} = \{nh\}$ and $Post_{AH} = \{\}$.

In the following, we assume that if the action node is ticked at some time instant $t \in \mathbb{N}$, the corresponding action is always over at time instant $t + 1$. As a consequence, the execution semantics of control flow nodes shown in Figure 3 is modified considering that $childStatus$ can never be RUNNING. In sequence and fallback nodes it is assumed that children are ticked, if ever, at increasing time instants, i.e., if the control flow node is ticked at time $t \in \mathbb{N}$, then its first child is ticked at time t, the second at time $t + 1$, and so on. In the case of parallel nodes, we assume that all children are ticked at the same time, i.e., the time in which also the parallel node gets ticked; further, we assume to have only action nodes as children in parallel nodes. Given an action $a \in \mathcal{A}$ and some time instant $t \in \mathbb{N}$, we write a_t (resp. $\tilde{a}_t$) to denote that action a is ticked at time t and succeeds (resp. fails) at time $t + 1$. We call a_t and $\tilde{a}_t$ *instantiations* of action a at time t, and we speak of successful and failed action instantiations, respectively. Given a CBT $\mathcal{T}$, we define an *action instantiation sequence* of $\mathcal{T}$, denoted as $\sigma_{\mathcal{T}} = \langle u_{t_0}, u_{t_1}, u_{t_2}, \ldots \rangle$, as a sequence of actions instantiations, i..e., $u_t = a_t$ or $u_t = \tilde{a}_t$ for some action $a \in \mathcal{A}$ which may get ticked in $\mathcal{T}$ at some time $t \in \mathbb{N}$, where $t_i \leq t_{i+1}$ for all $i \in \mathbb{N}$. The set of all such sequences for a CBT $\mathcal{T}$ is denoted as $\Sigma_{\mathcal{T}}$ in the following.

Example 2: Action instantiation sequences in the CBT defined in Example 1 include, e.g., $\langle GK_0, FB_1, TB_2, FeB_2\rangle$, where action GK is executed first and succeeds; then action FB is executed and succeeds, and finally actions TB and $\widetilde{FeB}$ are initiated in parallel and succeed. Another one is $\langle \widetilde{GK}_0, AH_1\rangle$ where action GK fails and then action AH is tried and succeeds. On the other hand, $\langle AH_0\rangle$ is not an action instantiation sequence because action AH may never be ticked first, and $\langle GK_0, AH_1\rangle$ is also not allowable because, if GK_0 is successful AH cannot be invoked next.

To take into account pre- and post-conditions, we define a *history* as a mapping $h : \mathbb{N} \to 2^{\mathcal{F}}$. Intuitively, at any point in time $t \in \mathbb{N}$, i.e., after t ticks have been generated, $h(t)$ is the set of fluents that hold at such point in time. For a generic $t \in \mathbb{N}^+$, we call $h(t)$ *state*, and $h(0)$ is the *initial state*. Given some history h and some instant $t \in \mathbb{N}$, we say that action $a \in \mathcal{A}$ is *executable* in a state $h(t)$, denoted $h(t) \models a$, exactly when, for all fluents $p \in \mathcal{F}$ such that $p \in Pre_a$, it is also the case that $p \in h(t)$, and for all fluents $q \in \mathcal{F}$ such that $\overline{q} \in Pre_a$, it is also the case that $q \notin h(t)$. Intuitively, an action is executable in state $h(t)$ exactly when its pre-conditions hold in that state. We extend the notation to action instantiations, and we write $h(t) \models u_t$ with $u_t = a_t$ or $u_t = \tilde{a}_t$ for some action $a \in \mathcal{A}$ as long as a is executable in state $h(t)$. Given a CBT $\mathcal{T}$, we say that a history h *supports* an action instantiation sequence $\sigma_{\mathcal{T}} = \{u_{t_0}, u_{t_1}, u_{t_2}, \ldots\}$, denoted with $h \models \sigma_{\mathcal{T}}$, exactly when for all $u_t \in \sigma_{\mathcal{T}}$:

- $h(t) \models u_t$ (every action instantiation must be executable at the time in which the corresponding action might get ticked in $\mathcal{T}$).
- If $u_t = a_t$, then for all $p \in Post_a$ it must be $p \in h(t+1)$, and for all $\overline{q} \in Post_a$ it must be $q \notin h(t+1)$; for all other $p \in \mathcal{F} \setminus Post_a$ it must be $p \in h(t+1)$ (resp. $p \notin h(t+1)$) exactly when $p \in h(t)$ (resp. $p \notin h(t)$) — as in STRIPS/PDDL planning, if an action instantiation succeeds at time t, its post-conditions must hold at time $t+1$; all other fluents do not change.
- if $u_t = \tilde{a}_t$, then for all $p \in \mathcal{F}$ it must be $p \in h(t+1)$ (resp. $p \notin h(t+1)$) exactly when $p \in h(t)$ (resp. $p \notin h(t)$) — if an action instantiation fails then nothing changes.

A CBT $\mathcal{T}$ is *executable* exactly when, for all action instantiation sequences $\sigma_{\mathcal{T}} \in \Sigma_{\mathcal{T}}$, there exists h such that $h \models \sigma_{\mathcal{T}}$.

Example 3: Consider again the BT represented in Figure 1 decorated into a CBT as shown in Example 1. The CBT is not executable because there exists at least one action instantiation sequence which is not supported by any history. To see this, consider the action instantiation sequence $\sigma = \langle GK_0, \widetilde{FB}_1, AH_2\rangle$; any history h supporting σ must be such that $nh \in h(0)$, $rk, bv \in h(1)$ and $nh \in h(2)$ to satisfy all the pre-conditions when actions are instantiated; however, it is also the case that $nh \notin h(1)$, since GK is successful at time 0; furthermore, $nh \notin h(2)$, given that no succesful action is executed to change the value of nh, which invalidates the execution of AH_2.

IV. Encoding executability

Below, we will consider CBTs whose longest plan has length equal to N. This can be done without loss of generality because CBTs (or BT) can represent finitely many plans with a fixed maximum length. Given a CBT $\mathcal{T}$, we want to obtain an encoding in propositional logic that is satisfiable exactly when there exists $\sigma_{\mathcal{T}} \in \Sigma_{\mathcal{T}}$ that is supported by no history. Further, we want that each model of the encoding corresponds to such an action instantiation sequence. To obtain such encoding, we have to create two propositional logic formulas:

- $\eta^{\Sigma'}$ whose models correspond to $\sigma' \in \Sigma'$, where Σ' is the set of action instantiation sequences $\sigma' = \langle u_{t_o}, u_{t_1}, u_{t_2}, \ldots\rangle$ with $u_t = a_t$ or $u_t = \tilde{a}_t$ for some $a \in \mathcal{A}$ such that there exists no $h : h \models \sigma'$, and
- $\eta^{\Sigma_{\mathcal{T}}}$ whose models correspond to $\sigma_{\mathcal{T}} \in \Sigma_{\mathcal{T}}$.

With these assumptions, the models of $\eta^{\Sigma'} \wedge \eta^{\Sigma_{\mathcal{T}}}$ correspond to the action instantiation sequences belonging to $\Sigma' \cap \Sigma_{\mathcal{T}}$. Hence, if the encoding is satisfiable, we have $\Sigma' \cap \Sigma_{\mathcal{T}} \neq \emptyset$.

A. Encoding of Σ'

The encoding $\eta^{\Sigma'}$ is the conjunction of three elements: for every $a \in \mathcal{A}$ and $i = 0, \ldots, N-1$

$$a_i \rightarrow \bigwedge\{c_{i+1} \mid c \in Post_a\};$$

for every $c \in \mathcal{C}$ and $i = 0, \ldots, N-1$

$$(\neg c_i \wedge c_{i+1}) \rightarrow \bigvee\{a_i \mid c \in Post_a\};$$

and for every $a \in \mathcal{A}$ and $i = 0, \ldots, N-1$ the disjunction of

$$a_i \wedge \big(\bigvee\{\neg c_i \mid c \in Pre_a\}\big).$$

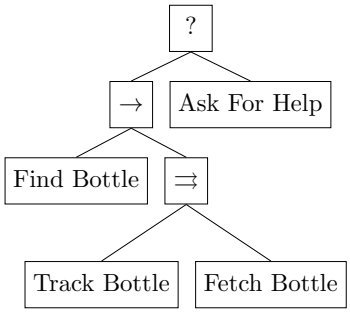
The above states that *(i)* if an action is instantiated, then at the next time step its post-conditions must hold, *(ii)* if a condition, that does not hold at time step t holds at time step $t+1$ then at least one action having it as post-condition must succeed at t, *(iii)* every action instantiation sequence must contain at least an action instantiation $u_t = a_t$ such that there exists no $h : h(t) \models u_T$. Computing the encoding $\eta^{\Sigma'}$ has time complexity $O(|\mathcal{A}|) + O(|\mathcal{C}|)$.

Example 4: Consider the CBT in Example 1, we will now show its encoding $\eta^{\Sigma'}$. First, we notice that $N = 4$. Then, we write the formulas below for $i = 0, \ldots, 3$ and we conjoin them:

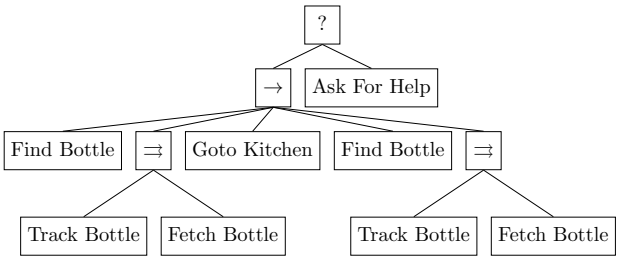
$$\begin{aligned}
&(GK_i \rightarrow (rk_{i+1} \wedge bv_{i+1} \wedge \neg nh_{i+1})) \wedge \\
&(FB_i \rightarrow bl_{i+1}) \wedge (FeB_i \rightarrow bf_{i+1}) \wedge \\
&((\neg rk_i \wedge rk_{i+1}) \rightarrow GK_i) \wedge ((\neg bv_i \wedge bv_{i+1}) \rightarrow GK_i) \wedge \\
&((nh_i \wedge \neg nh_{i+1}) \rightarrow GK_i) \wedge ((\neg bl_i \wedge bl_{i+1}) \rightarrow FB_i) \wedge \\
&((\neg bf_i \wedge bf_{i+1}) \rightarrow FeB_i) \wedge \\
&((rk_i \wedge \neg rk_{i+1}) \rightarrow \bot) \wedge ((bv_i \wedge \neg bv_{i+1}) \rightarrow \bot) \wedge \\
&((\neg nh_i \wedge nh_{i+1}) \rightarrow \bot) \wedge ((bl_i \wedge \neg bl_{i+1}) \rightarrow \bot) \wedge \\
&((bf_i \wedge \neg bf_{i+1}) \rightarrow \bot)
\end{aligned}$$

The formula obtained is conjoined with the disjunction of:

$$\big((GK_i \wedge \neg nh_i) \vee (AH_i \wedge \neg nh_i) \vee (FB_i \wedge (\neg rk_i \vee \neg bv_i)) \vee$$



(a) BT encoding the task for Experiment 1



(b) BT encoding the task for Experiment 3

Fig. 4. BTs encoding the tasks for Experiment 1 and 3 (Experiment 2 is the same as Figure 1).

$(TB_i \wedge (\neg rk_i \vee \neg bl_i)) \vee (FeB_i \wedge (\neg rk_i \vee \neg bl_i))$
for $i = 0, \ldots, 3$.

B. Encoding of $\Sigma_\mathcal{T}$

The encoding $\eta^{\Sigma_\mathcal{T}}$ can be obtained as follows:

1) Assign the time step t to each node.
2) Assign a unique identifier i to each node.
3) Conjoin the formulas below:

$$\bigwedge\{\neg b_t \mid b \in \mathcal{A} \setminus C\}$$

for every parallel node having children encoding the set of actions C at time t, together with

$$\bigwedge\{\neg b_t \mid b \in \mathcal{A} \setminus a\}$$

for every action node (not child of a parallel) encoding an action a at time t or with

$$\bigwedge\{\neg b_t \mid b \in \mathcal{A}\}$$

for every condition node (not child of a parallel) encoding a condition that must hold at time t.

4) For every node i, define two new variables: *(i)* η_i^{succ}, modeling the success of i, *(ii)* η_i^{try}, modeling the execution of i. For every node i of the CBT, we create a new formula given by the conjunction of $(\eta_i^{succ} \rightarrow \eta_i^{try})$, which is conjoined with the conjunction of $(\eta_i^{succ} \leftrightarrow x_t)$ for every execution node i representing either an action x at t or a condition x that must hold at t.
5) Inductively define the encoding of the plans associated to a successful CBT.
   - for every execution node i: η_i^{succ}
   - for every parallel node p with children $i = 1, \ldots, K$:

$$\big(\eta_p^{succ} \leftrightarrow (\eta_i^{succ} \wedge \ldots \wedge \eta_{i+K}^{succ})\big) \wedge \big(\eta_p^{try} \leftrightarrow (\eta_i^{try} \vee \ldots \vee \eta_{i+K}^{try})\big)$$

   - for every sequence node s with children $i = 1, \ldots, K$:

$$(\eta_s^{succ} \leftrightarrow (\eta_i^{succ} \wedge \ldots \wedge \eta_{i+K}^{succ})) \wedge (\eta_s^{try} \leftrightarrow \eta_i^{try}) \wedge \big((\eta_{i+K}^{try} \rightarrow \eta_{i+K-1}^{succ}) \wedge \ldots \wedge (\eta_{i+1}^{try} \rightarrow \eta_i^{succ})\big)$$

   - for every fallback node f with children $i = 1, \ldots, K$:

$$(\eta_f^{succ} \leftrightarrow (\eta_i^{succ} \wedge \neg\eta_{i+1}^{succ} \wedge \ldots \neg\eta_{i+K}^{succ}) \vee \ldots \vee (\eta_{i+K}^{succ} \wedge \neg\eta_i^{succ} \wedge \ldots \neg\eta_{i+K-1}^{succ})) \wedge (\eta_f^{try} \leftrightarrow \eta_i^{try}) \wedge \big((\eta_{i+K}^{try} \rightarrow \neg\eta_{i+K-1}^{succ}) \wedge \ldots \wedge (\eta_{i+1}^{try} \rightarrow \neg\eta_i^{succ})\big)$$

   - for the root node r with child c:

$$(\eta_r^{try} \leftrightarrow \eta_c^{succ}) \wedge \eta_r^{try}$$

Computing the encoding $\eta^{\Sigma_\mathcal{T}}$ requires polynomial time in the worst case, because encoding fallback nodes requires a quadratic number of symbols to be introduced. Hence the overall time complexity is $O(M^2)$ where M is the number of nodes of the BT. Since $M > |\mathcal{A}|$, the overall time complexity is $O(M^2) + O(|\mathcal{C}|)$.

Example 5: Continuing Example 4, the the set of all actions sequences represented by the CBT is given by the conjunction of:

$$\begin{aligned}
&(\neg FB_0 \wedge \neg TB_0 \wedge \neg FeB_0 \wedge \neg AH_0) \wedge \\
&(\neg GK_1 \wedge \neg TB_1 \wedge \neg FeB_1 \wedge \neg AH_1) \wedge \\
&(\neg GK_2 \wedge \neg FB_2 \wedge \neg AH_2) \wedge \\
&(\neg GK_3 \wedge \neg FB_3 \wedge \neg TB_3 \wedge \neg FeB_3) \wedge \\
&(\eta_1^{succ} \leftrightarrow GK_0) \wedge (\eta_2^{succ} \leftrightarrow FB_1) \wedge \\
&(\eta_3^{succ} \leftrightarrow TB_2) \wedge (\eta_4^{succ} \leftrightarrow FeB_2) \wedge \\
&(\eta_1^{succ} \rightarrow \eta_1^{try}) \wedge (\eta_2^{succ} \rightarrow \eta_2^{try}) \wedge (\eta_3^{succ} \rightarrow \eta_3^{try}) \wedge \\
&(\eta_4^{succ} \rightarrow \eta_4^{try}) \wedge (\eta_5^{succ} \rightarrow \eta_5^{try}) \wedge (\eta_6^{succ} \rightarrow \eta_6^{try}) \wedge \\
&(\eta_7^{succ} \rightarrow \eta_7^{try}) \wedge (\eta_8^{succ} \rightarrow \eta_8^{try}) \wedge \\
&\big(\eta_5^{succ} \leftrightarrow (\eta_3^{succ} \wedge \eta_4^{succ})\big) \wedge \big(\eta_5^{try} \leftrightarrow (\eta_3^{try} \vee \eta_4^{try})\big) \wedge \\
&\big(\eta_6^{succ} \leftrightarrow (\eta_1^{succ} \wedge \eta_2^{succ} \wedge \eta_5^{succ})\big) \wedge \\
&(\eta_5^{try} \rightarrow \eta_2^{succ}) \wedge (\eta_2^{try} \rightarrow \eta_1^{succ}) \wedge
\end{aligned}$$

$$(\eta_7^{succ} \leftrightarrow AH_3) \wedge$$
$$\big(\eta_8^{succ} \leftrightarrow ((\eta_6^{succ} \wedge \neg\eta_7^{succ}) \vee (\neg\eta_6^{succ} \wedge \eta_7^{succ}))\big) \wedge$$
$$(\eta_7^{try} \rightarrow \neg\eta_6^{succ}) \wedge$$
$$(\eta_r^{try} \leftrightarrow \eta_8^{succ}) \wedge \eta_r^{try}$$

Let us assume that we want to check whether there non-executable plans when the robot is initially near the human agent, while all other conditions do not hold. Given this initial condition, we have that our encoding is satisfiable and it has four possible models, associated with the following action instantiation sequences:

1) $\langle GK_0, \widetilde{FB}_1, AH_2 \rangle$
2) $\langle GK_0, FB_1, \widetilde{TB}_2, FeB_2, AH_3 \rangle$
3) $\langle GK_0, FB_1, TB_2, \widetilde{FeB}_2, AH_3 \rangle$
4) $\langle GK_0, FB_1, \widetilde{TB}_2, \widetilde{FeB}_2, AH_3 \rangle$

V. Experimental scenarios

In this section, we show experiments wherein we run our tool in some scenarios from the CARVE project. In Experiment 1, we show an example of an executable CBT, whereas in Experiment 2, we show an example of a non-executable CBT. In Experiment 3, we consider an example of a CBT that is executable only if a condition of the environment holds. Finally, in Experiment 4 we consider a CBT larger than those used in previous experiments to assess practical feasibility. We wish to point out that the generation of the above encodings and their satisfiability check always took less than 100ms of CPU time with our tool as can be seen in Table I.[3] All the tests have been conducted on a macOS High Sierra (10.13.6) mounting an Intel Core i7 (1,7 GHz) and 8GB RAM. Satisfiability checking was performed with `limboole 1.1`[4] (with `picoSAT` as backend, compiled with `gcc 4.2.1`).

Experiment 1: In this experiment we use the CBT depicted in Figure 4(a), while the pre- and post-conditions of each action are defined as in Example 1. Different initial conditions lead us to two different scenarios: *(i)* As in Figure 5(a), the robot is in the kitchen and near a human, while the bottle is neither fetched nor located, but it is visible. The initial state is described by the following set of propositions:

$$\{nh_0, rk_0, \neg bf_0, \neg bl_0, bv_0\}$$

By executing this CBT, the robot finds the bottle and fetches it. *(ii)* As in Figure 5(b), the robot is in the kitchen and near a human, while the the bottle is neither fetched nor located, and it is not even visible. The initial state is described by the following set of propositions:

$$\{nh_0, rk_0, \neg bf_0, \neg bl_0, \neg bv_0\}$$

By executing the CBT depicted in Figure 4(a), the robot cannot find the bottle and asks for help.

[3]Our tool does not support parallel nodes with control nodes as children, thus we report just the time for satisfiability checking in Experiment 4.

[4]`https://github.com/alescode/limboole`

TABLE I

Time to check executability.

Experiment	*time*
1	91 ms
2	92 ms
3	96 ms
4	12 ms

TABLE II

Conditions for some actions nodes in the CBT of Figure 6.

Node	**Precondition**	**Postcondition**
Compute Pre-grasp Pose	Pre- grasp Pose Exists	Pre-grasp Pose Computed
Goto Pre-grasp Pose	Pre-grasp Pose Computed	Robot at Pre-grasp Pose
Locate Bottle	Bottle Visible	Bottle Located with High Confidence
Grasp Bottle	Bottle Located with High Confidence	Bottle Fetched

Hence, if we impose the initial conditions to be $\{nh_0, rk_0, \neg bf_0, \neg bl_0\}$, we obtain that our encoding is unsatisfiable, no matter the value of bv_0. All plans encoded in the CBT are executable.

Experiment 2: The robot is asked to fetch a bottle in the kitchen. The policy of the robot is the same as the one used for the examples and hence is encoded by the CBT in Figure 1. The pre- and post-conditions of each action are defined as in the previous experiment. The initial state is depicted in Figure 5(c), the robot is in the living room, is near a human, the bottle is not visible, not fetched and not located. Thus, the initial state is described by the following set of propositions:

$$\{nh_0, \neg rk_0, \neg bv_0, \neg bl_0, \neg bf_0\}$$

Executing this CBT, the robot goes into the kitchen to fetch the bottle (Figure 5(d)). When *Find Bottle* fails — due to the bottle being partially visible as the chair is on the line of robot's sight —, the action *Ask for Help* is not executable anymore. Given the initial conditions, there exists a plan that is never executable. Our encoding will be satisfiable and will have models corresponding to the action instantiation sequences given in Example 5.

Experiment 3: The robot is asked to fetch a bottle in the kitchen first, and then another one in the living room afterwards. Both bottles cannot be located because the floor is not in the robot's visual field. The policy of the robot is encoded by the BT in Figure 4(b), while the pre- and post-conditions are defined as in the previous experiments. The initial state is depicted in Figure 5(e), the robot is in the living room, is near a human, the bottle is not visible, not fetched, and not located. The initial state is hence the same as in Experiment 2. When we execute this CBT, if the robot fails to locate the bottle in the living room (*Find Bottle* fails), the CBT can be completed because *Ask For Help* succeds (the human is near the robot). However, if the robot reaches the kitchen (as in Figure 5(f)), then the BT fails because the bottle cannot be located, and the human is not nearby (both *Find Bottle* and *Ask For Help* fail). The BT is not executable, and our encoding is satisfiable.

Experiment 4: In Experiment 4, we show an example of a CBT which is larger than the ones in previous experiments: in Figure 6 we show the corresponding BT. The pre- and

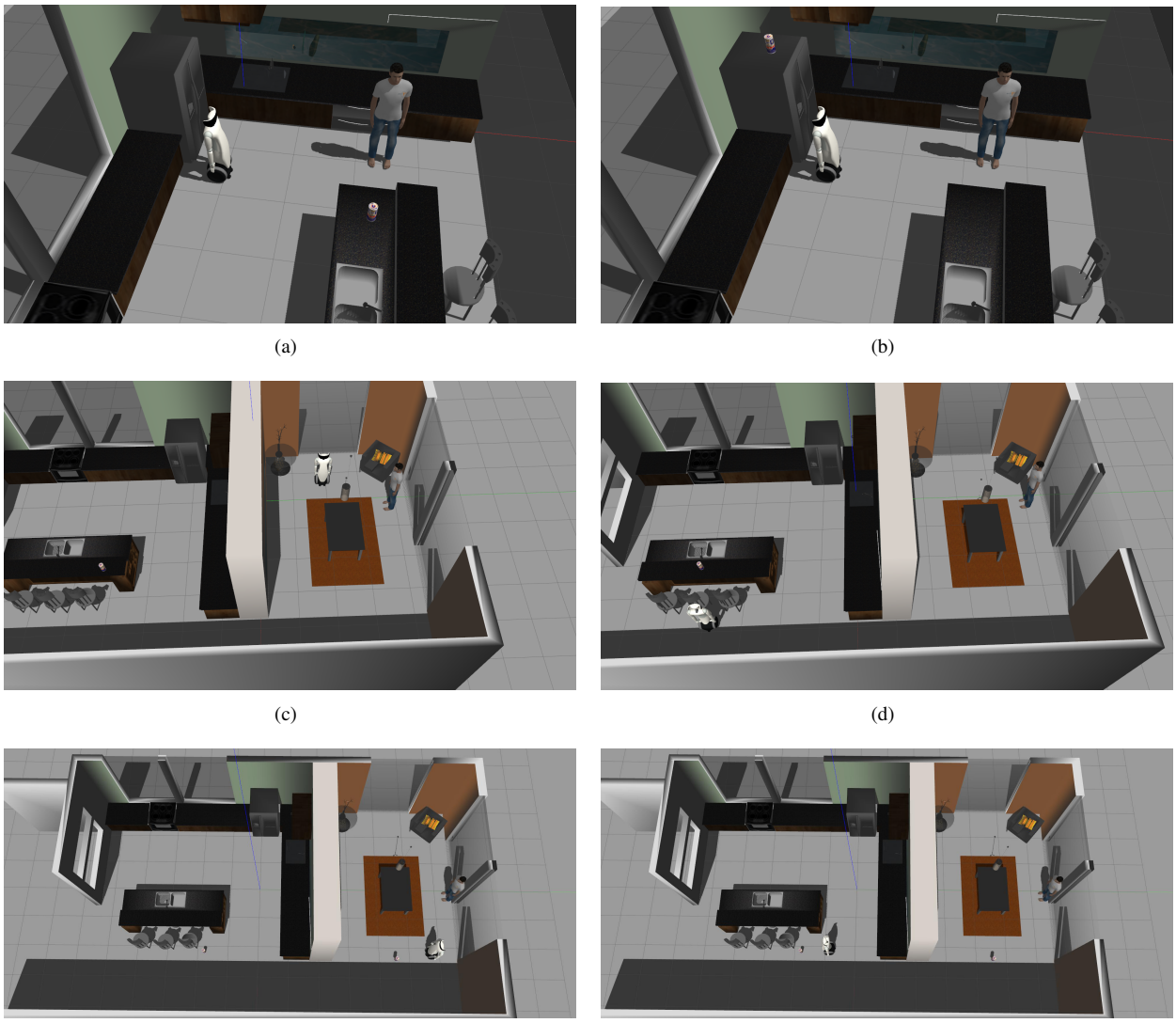


Fig. 5. Scenarios. 5(a) Initial state for Experiment 1, Scenario 1. 5(b) Initial state for Experiment 1, Scenario 2. 5(c) Initial state for Experiment 2. 5(d) Experiment 2: action *Find Bottle* failing because the bottle is partially occluded by the chair. 5(e) Initial state for Experiment 3. 5(f) Experiment 3: *Fetch Bottle* failing because the bottle is not visible.

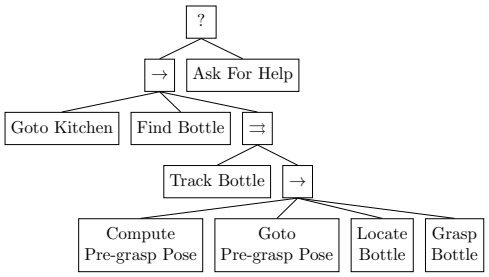


Fig. 6. CBT encoding the task of Scenario 4

post-conditions of the actions {Goto Kitchen, Find Bottle, Track Bottle and Ask for Help} are defined as in the previous experiments, while the pre- and post-conditions of the actions {Compute Pre-grasp Position, Goto Pre-grasp Pose, Locate Bottle and Grasp Bottle} are listed in Table II. In the initial state we have that the only two conditions satisfied are: Robot Near Human and Pre-grasp Pose Exists. As it can be noticed, here the assumption that a parallel node has only action nodes as children is dropped. Such assumption was added to make the discussion easier to follow. Nevertheless, it is possible to relax it and assume that, given a parallel node, all its children always define instantiation sequences of the same length. When a CBT does not respect such assumption, the problem can still be circumvented by adding *no action* nodes as necessary, i.e., action nodes which are always successful and last for 1 tick without changing fluents. Executing this CBT, the robot can go to the kitchen (hence can be far from the human) and any of the actions can fail there. Hence our encoding is satisfiable.

VI. Conclusions

In this paper we have introduced CBTs, a new formalism to annotate BTs with action conditions in the style of STRIPS/PDDL planning. We have shown how to encode executability of CBTs into propositional formulas in an efficient way — polynomial in the size of the CBT. Finally, we have shown that computing executability of CBTs is possible by leveraging current stateo-of-the-art SAT solvers, tackling scenarios of practical interest in reasonable time. Further developments along this line of research will include tackling more complex scenarios to assess the limits of scalability and relating this approach to other formal-based techniques, e.g., contract-bases analysis and monitoring.

VII. Acknowledgements

This work was carried out in the context of the CARVE project, which has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732410, in the form of financial support to third parties of the RobMoSys project. Eleonora Giunchiglia is supported by the EPSRC, under grant EP/M508111/1, and the Oxford-DeepMind Graduate Scholarship.

References

[1] A. J. Champandard, "Understanding behavior trees," *AiGameDev.com*, vol. 6, 2007.

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

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

[4] A. Klöckner, "Behavior trees with stateful tasks," in *Advances in Aerospace Guidance, Navigation and Control*. Springer, 2015, pp. 509–519.

[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] 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.

[7] 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.

[8] M. Colledanchise, D. Almeida, and P. Ögren, "Towards blended reactive planning and acting using behavior trees," in *IEEE International Conference on Robotics and Automation (ICRA)*, May 2019.

[9] M. Colledanchise and P. Ögren, *Behavior Trees in Robotics and AI: An Introduction*. CRC Press, 2018.

[10] 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)*, October 2016.

[11] J. A. Bagnell, F. Cavalcanti, L. Cui, T. Galluzzo, M. Hebert, M. Kazemi, M. Klingensmith, J. Libby, T. Y. Liu, N. S. Pollard, M. Pivtoraiko, J. Valois, and R. Zhu, "An integrated system for autonomous robotics manipulation," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2012, Vilamoura, Algarve, Portugal, October 7-12, 2012*, 2012, pp. 2955–2962.

[12] R. E. Fikes and N. J. Nilsson, "Strips: A new approach to the application of theorem proving to problem solving," *Artificial intelligence*, vol. 2, no. 3-4, pp. 189–208, 1971.

[13] M. Fox and D. Long, "Pddl2. 1: An extension to pddl for expressing temporal planning domains," *Journal of artificial intelligence research*, vol. 20, pp. 61–124, 2003.

[14] J. A. Ambros-Ingerson and S. Steel, "Integrating planning, execution and monitoring." in *AAAI*, vol. 88, 1988, pp. 21–26.

[15] S. A. Cook, "The complexity of theorem-proving procedures," in *Proceedings of the third annual ACM symposium on Theory of computing*. ACM, 1971, pp. 151–158.

[16] A. Biere, M. Heule, and H. van Maaren, *Handbook of satisfiability*. IOS press, 2009, vol. 185.

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

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

[19] F. Rovida, B. Grossmann, and V. Krüger, "Extended behavior trees for quick definition of flexible robotic tasks," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 6793–6800.

[20] A. Parmiggiani, L. Fiorio, A. Scalzo, A. V. Sureshbabu, M. Randazzo, M. Maggiali, U. Pattacini, H. Lehmann, V. Tikhanoff, D. Domenichelli, *et al.*, "The design and validation of the r1 personal humanoid," in *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*. IEEE, 2017, pp. 674–680.