Computing the Initial Requirements in Conditioned Behavior Trees

Eleonora Giunchiglia

DIBRIS, Università degli Studi di Genova, Italy
eleonora.giunchiglia@icloud.com

Abstract. Engineered multi agent systems are complex enough to require methods able to represent their global policies in a detailed yet intuitive way. Behavior Trees are widely adopted to represent agent policies in gaming industry where modularity and ease of specification are among the main requirements. However, their usage is mostly informal and restricted to single agent policies. In this paper we propose an extension of Behavior Trees, called Conditioned Behavior Trees, to deal with formal pre- and post- conditions related to action execution in a multi agent context. For such trees, we further describe an encoding which enables the computation of initial requirements, i.e., the set of conditions under which at least one of the sequences of actions encoded by the policy is bound to be executable.

1 Introduction

Engineered multi agent systems (MAS) are complex enough to require methods able to represent their global policies in a detailed yet intuitive way. To this purpose, many visual tools (e.g., [4]) have been developed in the past years, nevertheless we believe that Behavior Trees, given their intuitiveness and compactness, could represent a viable alternative to such tools. Behavior Trees (BTs) [2] are very popular in the computer gaming industry (see e.g., [1, 3]) thanks to their exploitation of modularity of tasks and their intuitive aspect. Nevertheless, BTs have been mostly used to represent policies for single agents and, moreover, a formal definition has never been presented in their literature.

In this paper, we propose an extension of BTs, called Conditioned Behavior Trees (CBTs), which are able not only to represent global policies for MAS but also to deal with actions which are subject to pre- and post-conditions. Further, we show how, given a CBT, it is possible to compute in polynomial time a propositional formula whose models correspond to the initial requirements of the CBT, i.e., the set of conditions under which at least one of the sequences of actions encoded by the policy is bound to be executable.

The paper is structured as follows: in Section 2 we revise the main concepts about BTs, in Section 3 we describe CBTs and then, in Section 4 we show how to perform the automatic computation of the initial requirements providing an example of such computation in Section 5. Finally, we conclude the paper with the conclusions and future work in Section 6.

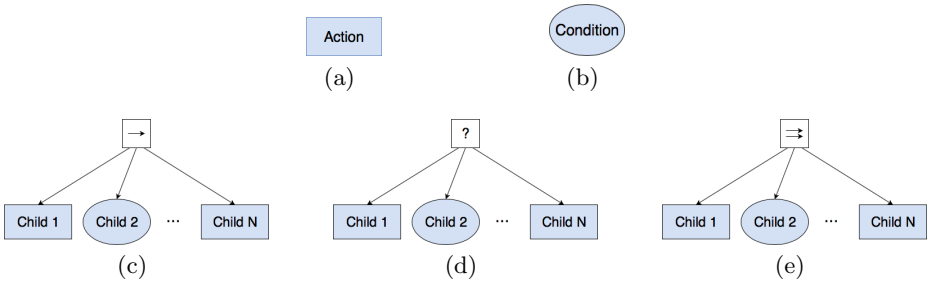

Fig. 1. Graphical representation of each type of node. Fig. 1(a) action node, Fig. 1(b) condition node, Fig. 1(c) sequence node and children, Fig. 1(d) fallback node and children, Fig. 1(e) parallel node and children.

2 Behavior Trees

Following [2], informally a BT is a directed rooted tree in which the nodes are classified as root, control flow nodes or execution nodes; control flow nodes are always internal nodes, and the execution nodes are always leaves. The execution of every BT starts from the root, which sends ticks with a given frequency to its children which can return *Running*, if its execution is not yet completed, *Success*, if it has achieved its goal, and *Failure* otherwise. There are two types of execution nodes (action and condition) and four types of control flow nodes (sequence, fallback, parallel and decorator):

1. Action node: when it receives a tick from its parent it returns *Success* if the action is successfully completed, *Failure* if the action cannot be completed and *Running* otherwise.
2. Condition node: when it receives a tick from its parent, it returns *Success* if the condition is satisfied and *Failure* otherwise. It cannot return *Running*.
3. Sequence node($\rightarrow$): when it receives a tick from its parent it sends it to its children in succession, returning *Failure* (*Running*) as soon as one of them returns *Failure* (*Running*), and *Success* only when all the children have already returned *Success*.
4. Fallback node(?): when it receives a tick from its parent it sends it to its children in succession, returning *Success* (*Running*) as one of them returns *Success* (*Running*), and *Failure* only if all the children return *Failure*.
5. Parallel node($\rightrightarrows$): when it receives a tick from its parent it sends it to its N children in parallel and returns *Success* if a given number M of children returns *Success*, *Failure* if $N - M + 1$ return *Failure* and *Running* otherwise. In this paper we suppose that a parallel returns *Success* if all N children return *Success*, and that all its children are action nodes.
6. Decorator node: special node with a single child which changes the child's outcome according to a policy defined by the user. Since it is user defined, we will not focus on it in this paper.

A representation of each type of node is given in Figure 1. If we define a plan as a sequence of sets of actions, then we understand that the policy defined by the BT is the set of plans that are compliant with the BT itself, and which can be easily derived from the rules we have given above.

3 Conditioned Behavior Trees

Intuitively, a CBT extends classical BTs because for each action it specifies: the agent that performs the action, the action performed by the agent, the set of preconditions which must be satisfied in order to perform the action, and the set of postconditions that are necessarily satisfied if the action is successful.

More formally, we define a CBT over three sets:

1. the finite set of ids $\mathcal{AG}$ representing the agents belonging to the MAS,
2. the finite set of ids $\mathcal{A}$ representing the actions that each agent may perform,
3. the finite set of literals $\mathcal{C}$ used to describe the state of an agent or the state of the environment in which the agents operate. We call $\mathcal{C}^+$ the set of positive literals in $\mathcal{C}$ and $\mathcal{C}^-$ the set of negative literals in $\mathcal{C}$.

Hence, given the sets $\mathcal{A}$, $\mathcal{C}$ $\mathcal{AG}$ and the function $perform : \mathcal{AG} \rightarrow \mathcal{P}(\mathcal{A})$ such that given an agent $\alpha \in \mathcal{AG}$ it returns the set of actions α can perform, the designer can define a function *cond* that associates to each couple (α, a) with $\alpha \in \mathcal{AG}$ and $a \in perform(\alpha)$ a couple $(Pre,\ Post)$ in which:

1. $Pre \subset \mathcal{C}$ represents the set of preconditions, and
2. $Post \subset \mathcal{C}$ represents the set of postconditions.

In the following we will always deal with agent-action couples (α, a) with $a \in perform(\alpha)$. When modeling the CBT we make the following additional assumptions:

1. each action is instantaneous,
2. given a couple agent-action (α, a) it cannot be the case that $cond(\alpha, a) = (Pre, Post)$ such that there exist $c \in Post$ and $\neg c \in Post$,
3. if two couples (α, a) and (β, b) are children of the same parallel node, then it cannot be the case that $cond(\alpha, a) = (Pre_{\alpha:a}, Post_{\alpha:a})$ and $cond(\beta, b) = (Pre_{\beta:b}, Post_{\beta:b})$ such that there exist $c \in Post_{\alpha:a}$ and $\neg c \in Post_{\beta:b}$.

As can be seen in Figure 2, given an agent α and an action a performed by α, we can represent them in our CBT as an action node with on top the label "$\alpha : a$" and below the sets Pre and $Post$ such that $cond(\alpha, a) = (Pre,\ Post)$. Further, in CBTs, we model any condition node having condition $c \in \mathcal{C}$ by associating to it a dummy action a, $Pre = \{c\}$ and $Post = \emptyset$.

4 Computing the initial requirements

4.1 Building the state graph

Given the sets $\mathcal{AG}$, $\mathcal{A}$ and $\mathcal{C}$, we can build a state graph $\langle \mathcal{S}, \mathcal{T} \rangle$ which represents all the possible plans carried out by the agents in $\mathcal{AG}$ and in which:

- $\mathcal{S}$ represents the set all of possible states and it is computed as $\mathcal{S} = \mathcal{P}(\mathcal{C}^+)$.
- $\mathcal{T}$ represents the transition function $\mathcal{T} : \mathcal{S} \times \mathcal{P}(\mathcal{AG} \times \mathcal{A}) \rightarrow \mathcal{S}$ such that for each state $s \in \mathcal{S}$ and set of actions performed by a set of agents $A \in \mathcal{P}(\mathcal{AG} \times \mathcal{A})$ it returns a new state s'. Given s and A we will be able to perform the set of actions A if for every pre-condition c of every action a performed by the agent α such that $\alpha : a \in A$ we have that:
  - if $c \in \mathcal{C}^+$ then $c \in s$, and
  - if $c \in \mathcal{C}^-$ then $c \notin s$.

  Given s and A the new state s' will be equal to:

$$s' = s \cup \{c \mid c \in \mathcal{C}^+, c \in Post(\alpha : a), \alpha : a \in A\}$$
$$\setminus \{c \mid c \in \mathcal{C}^-, c \in Post(\alpha : a), \alpha : a \in A\}.$$

Given the above definitions, we can notice that a CBT always defines finite plans, and, hence, it is possible to build a representation of all these plans on the state graph as a propositional logic formula. As we show below, those representations can be obtained in polynomial time with respect to the number of nodes in the CBT, and give us the ability of computing the initial requirements necessary to successfully complete at least one of the plans associated to the CBT.

4.2 State Graph Plans Representation

Suppose that the longest sequence of set of actions has length equal to N. Given the above, and given the sets $\mathcal{A}, \mathcal{C}$, and $\mathcal{AG}$ we can then represent all the possible plans of length up to N defined by the state graph as a conjunction of the below formulas:

- for every $\alpha : a \in \mathcal{P}(\mathcal{AG} \times \mathcal{A})$ then $(\alpha : a)_i \rightarrow \bigwedge\{c_i \mid c \in Pre(\alpha : a)\}$, $(\alpha : a)_i \rightarrow \bigwedge\{c_{i+1} \mid c \in Post(\alpha : a)\}$,
- for every condition $c \in \mathcal{C}$ then $(\neg c_i \wedge c_{i+1}) \rightarrow \bigvee\{(\alpha : a)_i \mid c \in Post(\alpha : a)\}$

for $i = 0, \ldots, N-1$.

4.3 Behavior Tree Plans Representation

In order to compute the representation of the CBT plans we have to:

1. assign to each node its sequence length equal to: (1) 1 if the node is a parallel or execution node, (2) the maximum sequence length of its children if it is a fallback node, (3) the sum of the sequence lengths of its children if it is a sequence node. Then,
2. compute the CBT plans representation by calling the function GETPLANS in Algorithm 1, with first argument set to the CBT root and the second set to 0. The function writes for each node n the encoding of the plans associated to n and returns the time at which the next action has to start.

Algorithm 1 Get Propositional Formula from CBT

```
function performActions(B, t)
    write (⋀{α : a_t | (α : a) ∈ B} ⋀{¬(β : b_t) | (β : b) ∈ (AG × A) \ B})
function getPlans(node, current_time)
    no_ops_time = node.parent.seq_length - node.seq_length
    if node.parent is fallback then
        for (i=0; i < no_ops_time; i++) do
            performActions(∅, current_time+i)
        current_time += no_ops_time
    if node is execution_node then
        write performActions({node.agent : node.action}, current_time)
        return current_time + 1
    if node is sequence or root then
        write (
        for c in node.children do
            current_time = getPlans(c, current_time)
            write ∧
        write ⊤ )
        return current_time
    if node is fallback then
        write (
        for c in node.children do
            getPlans(c, current_time)
            write ∨
        write ⊥ )
        return current_time + node.seq_length
    if node is parallel then
        C = { c.agent : c. action | c ∈ node.children}
        performActions(C, current_time)
        return current_time +1
```

5 Example

To make more intuitive the explained method we build the representations associated to the CBT in Figure 2. In the example, first we compute the length of the longest sequence of actions associated to each node obtaining that nodes 1 and 2 have sequence length equal to 2 while nodes 3, 4 and 5 have sequence length equal to 1. Since the longest sequence has length $N = 2$, we have that for $i = 0, 1$ the below formulas are produced: the state graph plans representation are:

$$\alpha : PassObject_i \rightarrow \neg ObjectNear\beta_i \tag{1}$$

$$\alpha : PassObject_i \rightarrow ObjectNear\beta_{i+1} \tag{2}$$

$$\beta : UseObject_i \rightarrow ObjectNear\beta_i \wedge ObjectFunctioning_i \tag{3}$$

$$\beta : UseObject_i \rightarrow \top \tag{4}$$

$$(ObjectNear\beta_i \wedge \neg ObjectNear\beta_{i+1}) \rightarrow \bot \tag{5}$$

$$(\neg ObjectNear\beta_i \wedge ObjectNear\beta_{i+1}) \rightarrow \alpha : PassObject_i \tag{6}$$

$$(ObjectFunctioning_i \wedge \neg ObjectFunctioning_{i+1}) \rightarrow \bot \tag{7}$$

$$(\neg ObjectFunctioning_i \wedge ObjectFunctioning_{i+1}) \rightarrow \bot \tag{8}$$

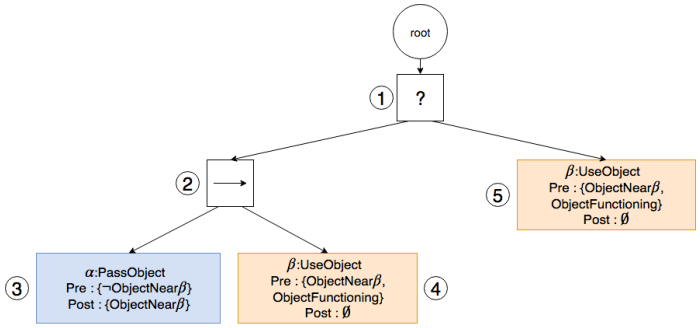

Fig. 2. Example of CBT.

while the CBT plans representation are:

$$\begin{aligned}((\alpha{:}PassObject_0 \wedge \neg\alpha{:}UseObject_0 \wedge \neg\beta{:}PassObject_0 \wedge \neg\beta{:}UseObject_0)\wedge\\ (\beta{:}UseObject_1 \wedge \neg\alpha{:}PassObject_1 \wedge \neg\alpha{:}UseObject_1 \wedge \neg\beta{:}UseObject_1))\vee\\ ((\neg\alpha{:}PassObject_0 \wedge \neg\alpha{:}UseObject_0 \wedge \neg\beta{:}PassObject_0 \wedge \neg\beta{:}UseObject_0)\wedge\\ (\alpha{:}UseObject_1 \wedge \neg\alpha{:}PassObject_1 \wedge \neg\beta{:}PassObject_1 \wedge \neg\beta{:}UseObject_1))\end{aligned}$$

If we check the satisfiability of the conjunction of the above formulas we get that the initial requirements are:

$$\{ObjectFunctioning_0\}$$

which correspond to the model of the equation above put in conjunction with (1)-(8).

6 Conclusions and Future Work

In this paper we have introduced a new type of BT, the Conditioned Behavior Tree, which can easily represent MAS and comes with a formal definition. Further, we have shown that, given a generic CBT, we can compute the initial requirements. In the future we would like to extend this work with more expressive logics in order to introduce time dependent pre and post conditions.

References

1. Champandard, A.J.: Understanding behavior trees. AiGameDev.com 6 (2007)
2. Colledanchise, M., Ögren, P.: Behavior trees in robotics and AI: an introduction. CoRR abs/1709.00084 (2017), http://arxiv.org/abs/1709.00084
3. Isla, D.: Halo 3-building a better battle. In: Game Developers Conference (2008)
4. Thangarajah, J., Padgham, L., Winikoff, M.: Prometheus design tool. In: 4th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005), July 25-29, 2005, Utrecht, The Netherlands. pp. 127–128 (2005), http://doi.acm.org/10.1145/1082473.1082817