Behavior Trees in Robot Control Systems

Petter Ögren,[1] and Christopher I. Sprague[1]

[1]KTH Royal Institute of Technology, Stockholm, Sweden, SE-10044; email: petter,sprague@kth.se

Annual Review of Control, Robotics, and Autonomous Systems 2022. 5:1–27

https://doi.org/10.1146/annurev-control-042920-095314

Copyright © 2022 by Annual Reviews.
All rights reserved

Keywords

behavior trees, modularity, hierarchical modularity, transparency, robustness, autonomous system, feedback, task switching
行为树，模块化，层次模块化，透明性，鲁棒性，自主系统，反馈，任务切换

Abstract

本文将从控制理论的角度探讨机器人行为树的研究领域。行为树的核心思想是利用模块化、层次结构和反馈，以应对多功能机器人控制系统的复杂性。模块化是一种众所周知的处理软件复杂性的方法，它允许在不需要详细了解整个系统的情况下，独立地开发、调试和扩展各个模块。由于机器人任务通常可以分解为层次化的子任务，因此这种模块的层次结构是自然而然的。最后，反馈控制是处理任何低级控制系统中不确定性和干扰的基本工具，但为了在更高级别启用反馈控制——即由一个模块决定执行哪个子模块——需要在模块接口中共享关于每个子模块的进度和适用性的信息。

我们将描述这三个概念如何在理论分析、实际设计，以及与控制理论和机器人学中其他思想的扩展与结合中发挥作用。

Contents

1. 引言

本节将描述为何模块化、层次结构和反馈在机器人控制系统中有用，以及这三种概念如何结合成一种称为行为树（BTs）的控制结构。

机器人硬件和软件的快速发展使机器人应用从结构化的工厂环境扩展到我们的家庭、街道和多样化的工作场所。在这些新环境中，机器人通常需要具备广泛的能力，包括通过在线软件更新添加更多功能的可能性。众所周知，增加软件功能会增加复杂性，而复杂性又会增加开发成本 (1)。同样众所周知，模块化是降低复杂性的关键原则。通过将系统划分为具有明确定义接口和功能的模块，每个模块都可以在不了解系统其余部分的情况下独立开发、测试和扩展。因此，有理由相信模块化——就其明确定义的接口和功能而言——也是机器人控制系统的重要属性。

模块化的自然扩展是层次模块化 (1)，其中模块可以包含子模块，子模块又可以包含子模块，以此类推。这种结构的合理性在于一个简单观察：当系统规模增大时，单层模块要么导致模块数量非常庞大，要么模块本身非常庞大。因此，如果模块可以以层次方式包含子模块，模块化的好处将更加明显。在机器人控制系统中，层次模块化还有一个额外的理由，即许多机器人任务本身可以自然地以层次方式划分为子任务，这一观察是层次任务网络（hierarchical task networks）等方法的基础 (2, 3)。例如，取回一个物品可能涉及移动到橱柜并打开它，而打开橱柜又可能涉及抓握把手，依此类推。

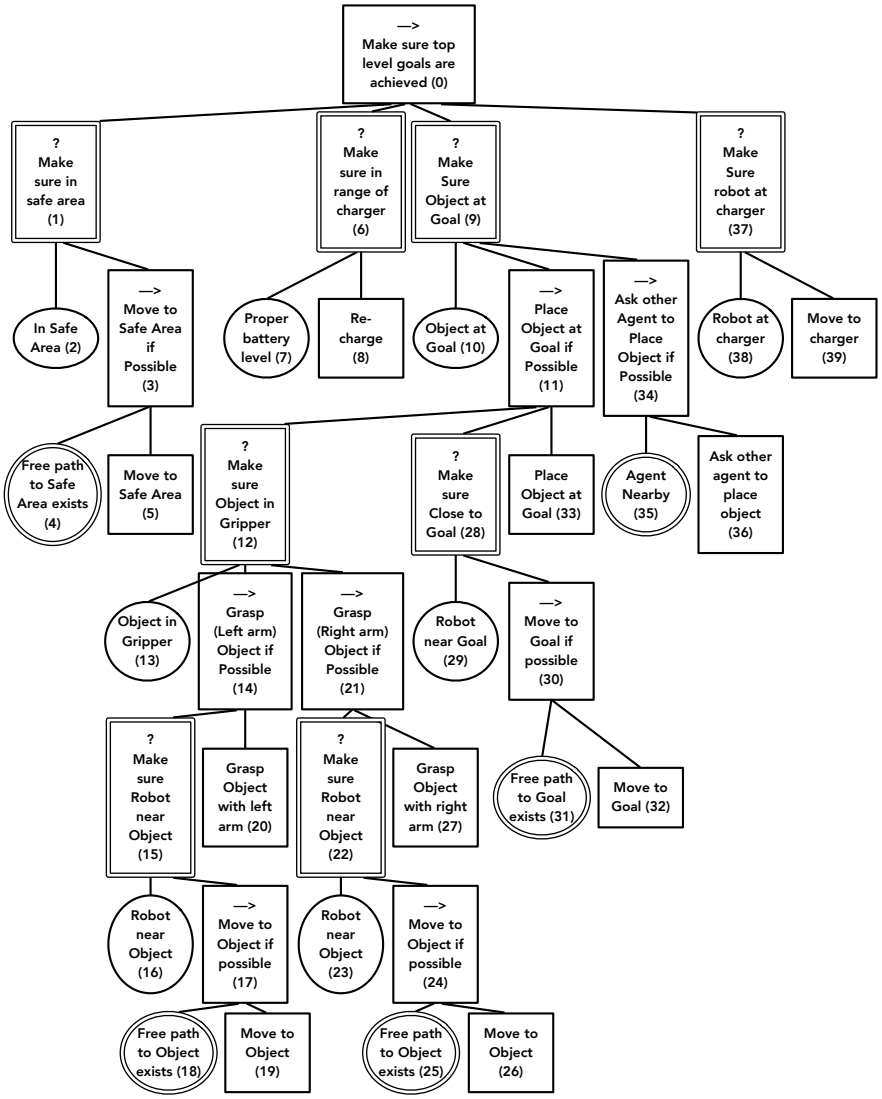


Figure 1: 一个移动操作臂的行为树。顶层目标位于最上方一行：*确保安全区域内 (1)*、*确保充电器范围内 (6)*、*确保物体到达目标 (9)* 和 *确保机器人位于充电器 (37)*，按优先级排序。如果在某一时刻执行动作 *移动到物体 (26)*，操作者可以通过读取沿向根节点分支上所有带双线框的模块，轻松理解为何选择该动作：*移动到物体 (26)*，为了 *确保机器人靠近物体 (22)*，（为了）*确保物体在夹持器中 (12)*，（为了）*确保物体到达目标 (9)*。双线框的含义将在第6节详细解释。

为了使控制系统模块化，我们将使实际的控制策略，即从状态到动作的映射，具有模块化。在机器人和控制的许多应用中，需要将一个控制策略由一组子策略组成。在自动驾驶汽车中，可能有停车、超车、保持车道、处理路口等子策略；而在移动操作臂中，可能有抓取、对接充电器、从 A 点移动到 B 点等子策略。

反馈或许是控制理论中最重要的原则，也是开环控制与闭环控制的区别所在。在开环控制中，一系列指令根据某种计划随时间执行，而在闭环控制中，发出的指令会根据从监测世界状态关键部分获得的当前信息不断调整。显然，经典的闭环控制应当在层次模块化机器人控制系统的最低层执行，但在两个层次之间应使用何种观测以允许一个模块利用反馈来决定执行哪个子模块，则尚不明确。我们稍后会回到这个问题，但现在仅指出，如果某个子模块未能实现其目标，我们不希望父模块只是以开环方式执行下一个子模块，而是基于前一个子模块失败的事实，利用反馈选择合适的子模块。

行为树（BTs）旨在将反馈与层次模块化设计结合起来。因此，模块应当捕获某些功能，这些功能可以组合成更大的模块，且所有层级模块间具有明确定义的接口。此外，关于执行的反馈应通过相同的接口向模块层级上传递。

行为树的正式定义见第 3节，这里我们做非正式描述。我们将上述讨论中的每个模块视为一个行为树（BT)。因此，复杂的行为树可以包含多个子行为树，如图 1所示，图中每个节点都是一个子行为树的根。所有行为树的接口（图中的连线）以函数调用及返回值的形式给出。当调用一个行为树时，它返回两个内容：首先是建议的控制动作，其次是用于应用反馈控制并决定执行哪个子行为树的信息。该信息或元数据，关于模块的执行和适用性，以三个符号之一表示，S（成功)、F（失败）和 R（运行中)。因此，如果抓取杯子的子模块返回成功，则可能会调用序列中的下一个子模块，比如抬起杯子。另一方面，如果子模块返回失败，则需要调用某种备用动作，例如尝试重新抓取杯子，或获取其姿态的更好传感器读数。最后，如果子模块返回运行中，可能更倾向于让执行继续一段时间。

此时我们注意到，基本上有三种根本原因会让你停止当前活动并开始新的活动。要么你成功并继续执行下一个动作，要么你失败，需要处理这一或多或少意外的事实，或者发生了一个外部事件，使当前动作变得不适当。想象一个机器人被指派去取回一个物体，如图 1所示。如果机器人正在抓取物体，当抓取成功时，它可能切换到移动动作。如果抓取失败，它可能尝试用另一只手臂。还有多种事件可能结束抓取物体的过程。例如，另一代理可能将物体放回适当位置（正面惊喜，我们完成了)，或者另一代理可能将物体移得更远（负面惊喜，我们需要重新靠近)，或者火警响起使整个建筑不安全（无关惊喜，我们需要离开建筑物)。

本文大纲如下。第 2节简要介绍行为树的历史，紧接着第 3节给出正式定义。随后，第 4节更详细地探讨模块化特性。第 5节分析收敛性问题，随后第 6节介绍设计原则。第 7节讨论安全保障及其与控制障碍函数的联系。接着，第 8节探讨行为树与可解释人工智能的关系，第 9节讨论其与强化学习的连接，第 10节涉及进化算法，第 11节涵盖规划。最后，第 12节给出结论、总结要点及未来重要问题。

2. 行为树的历史及其与有限状态机的关系

模块化层次控制结构的需求在机器人和电子游戏领域中是共通的。然而，机器人领域中诸如抓取和导航等底层能力本身就是研究领域，而在电子游戏虚拟世界中则较为简单。因此，游戏程序员比机器人专家更早开始整合大量底层能力，因此也较早经历了有限状态机（FSM)

以下所述的缺点。行为树（BTs）正是程序员针对这些缺点提出的解决方案。很难确定谁最先提出行为树，因为重要思想在部分文档化的研讨会、会议和博客文章中广泛传播。然而，Michael Mateas 和 Andrew Stern (4) 以及 Damian Isla (5) 的工作无疑是重要的里程碑。随后游戏人工智能社区持续发展，几年后第一篇关于行为树的期刊论文出现 (6)，紧接着机器人领域独立发表了首批行为树论文，见 (7) 和 (8)。注意，还有一种完全不同的工具也称为行为树，用于需求分析[1]。

如上所述，行为树部分是为改善有限状态机控制器的模块化而发展起来的。有限状态机，特别是层次有限状态机（HFSM）(9)，确实具备层次模块化机制。但关键问题在于，有限状态机的转换编码在模块（状态）内部，因此每个模块需了解其他模块的存在和能力，以及其自身上层模块的目的。这样，每个转换就在两个模块之间产生依赖，N 个模块可能有 N^2 条转换/依赖。相比之下，行为树模块只需知道自身是否成功。关于表达能力，文献 (10) 证明带有内部变量的行为树与有限状态机的表达能力相当。因此，就像两种通用编程语言的选择一样，二者的选择不在于能做什么，而在于哪个使设计过程更顺畅。关于行为树与有限状态机的关系的详细描述，以及行为树研究的广泛综述，可参见最新综述 (11) 和专著 (12)。

3. 行为树的定义

本节将对行为树（BTs）及其在离散时间和连续时间系统中的执行进行正式定义。该表述基于文献 (13, 14, 15)，旨在支持对行为树的控制理论分析[2]。

核心思想是将行为树形式化定义为控制器与用于提供执行反馈的元数据函数的组合。利用元数据，这些行为树可以层次化组合成更复杂的行为树，如图 1所示，因此称为行为树。

设系统状态为 $x \in X \subset \mathbb{R}^n$，系统动力学为 $\dot{x} = f(x,u)$ 或 $x_{t+1} = f(x_t,u)$，参见定义 2，其中 $u \in U \subset \mathbb{R}^k$。

Definition 1 (行为树)**.** *行为树 $\mathcal{T}_i$ 是一对*

$$\mathcal{T}_i = (u_i, r_i) \tag{1.}$$

其中 i 为索引，$u_i : X \to U$ 是在行为树执行时运行的控制器，$r_i : X \to \{\mathscr{R}, \mathscr{S}, \mathscr{F}\}$ 提供关于执行适用性和进度的元数据。

行为树可以通过层次组合其他行为树，利用下述的序列（Sequence）和回退（Fallback）算子构造，也可以通过直接指定 $u_i(x)$ 和 $r_i(x)$ 定义。

元数据 r_i 解释如下：*运行中* ($\mathscr{R}$)，*成功* ($\mathscr{S}$)，*失败* ($\mathscr{F}$)。设运行区域 (R_i)、成功区域 (S_i)

[1]同名不同概念：`https://en.wikipedia.org/wiki/Behavior_tree`

[2]还存在其他行为树的表述，包括带有内部节点记忆的版本，以及封装系统动力学执行的叶节点，从而允许两个叶节点并行执行，例如控制同一机器人上的不同电机，详见 (13, 16)。

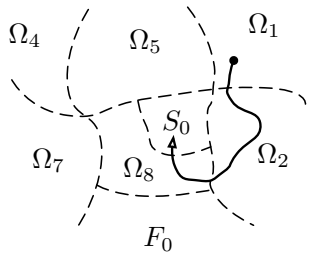


Figure 2: 状态空间 X 被划分为一组操作区域 Ω_i，以及全局成功区域 S_0 和失败区域 F_0。设计行为树时，我们期望状态达到 S_0 并避免 F_0。实线示意从 Ω_1 开始并最终进入 S_0 的执行路径。注意，每个子树的 Ω_i 并非由子树自身定义，而是依赖于邻近若干子树返回状态 $r_i(x)$ 的反馈，详见定义 11。

和失败区域 (F_i) 对应于状态空间的划分[3]，定义如下：

$$R_i = \{x \in X : r_i(x) = \mathscr{R}\},\ S_i = \{x \in X : r_i(x) = \mathscr{S}\},\ F_i = \{x \in X : r_i(x) = \mathscr{F}\}.$$

Definition 2. 假设行为树 $\mathscr{T}_i$ 是根节点，且不是其他行为树的子树，且 $x \in R_i$，系统按照 $\dot{x} = f(x, u_i(x))$ 或 $x_{t+1} = f(x_t, u_i(x))$ 演化，取决于系统是连续时间还是离散时间。

Remark 1. 若根行为树的状态 $x \in S_i \cup F_i$，则行为树已成功或失败，用户需采取适当动作，如关闭机器人或进入空闲模式。若不希望如此，可以设计一个额外的顶层行为树，当主行为树返回成功或失败时执行。若该顶层行为树始终返回运行中，则整体行为树满足 $S_i = F_i = \emptyset$。

行为树的执行因此可视为不连续动力系统 (17)，如图 2所示。在引理 8 中，我们将证明若行为树 $\mathscr{T}_0$ 由子树集合 $\{\mathscr{T}_i\}$ 组成，则状态空间 X 被划分为不同的所谓操作区域 Ω_i，使得当 $x \in \Omega_i$ 时，有 $u_0(x) = u_i(x)$，如图 2所示。因此，上述连续时间情况下，执行通常为不连续动力系统，存在关于解的存在性和唯一性的问题 (18)。深入探讨这些问题超出本文范围，故我们仅作如下假设。

Assumption 1. 行为树 $\mathscr{T}_i$ 的定义保证定义 *2*中的执行存在唯一解。

如上所述，行为树的主要目的是以模块化方式从简单控制器构建复杂控制器。行为树的组合有两种方式：序列组合和回退组合。

Definition 3. （行为树的序列组合） 两个或多个行为树可以用序列算子组合成更复杂的行为树，$\mathscr{T}_0 = \mathit{Sequence}(\mathscr{T}_1, \mathscr{T}_2)$. 则 r_0, u_0 定义如下：

$$\text{若} x \in S_1: \quad r_0(x) = r_2(x),\ u_0(x) = u_2(x) \tag{2.}$$

$$\text{否则}: \quad r_0(x) = r_1(x),\ u_0(x) = u_1(x) \tag{3.}$$

[3]本文中使用“划分”一词，即使某些集合可能为空。

$\mathscr{T}_1$ 和 $\mathscr{T}_2$ 称为 $\mathscr{T}_0$ 的子节点。注意执行 $\mathscr{T}_0$ 时，首先执行子节点 $\mathscr{T}_1$，只要其返回 运行中 或 失败（即 $x_k \notin S_1$），则继续执行 $\mathscr{T}_1$。序列中的第二个子节点 $\mathscr{T}_2$ 仅当第一个子节点返回 成功（$x_k \in S_1$）时执行。最终，序列节点 $\mathscr{T}_0$ 仅当所有子节点均成功（$x \in S_1 \cap S_2$）时返回 成功。

为方便表示，我们写作

$$\text{Sequence}(\mathscr{T}_1, \text{Sequence}(\mathscr{T}_2, \mathscr{T}_3)) = \text{Sequence}(\mathscr{T}_1, \mathscr{T}_2, \mathscr{T}_3), \tag{4.}$$

任意长度的序列组合类似。序列节点亦用符号（$\rightarrow$）表示，如图 1所示。

Remark 2 (为序列节点和回退节点命名)**.** 绘制行为树时，如图 *1*，符号 $\rightarrow$ 和 ? 用于表示序列和回退节点。但部分用户喜欢为每个节点起描述性名称，以提升可读性。我们认为这是有益的实践，类似于编程时为函数选择良好名称。为所有节点命名增强可读性，强调所有子树（包括单一叶节点）对父节点具有相同接口，详见定义 *1*，且便于结合软件 *GUI* 使用，可将子树折叠为单节点再展开。

合理命名子树的优势在图 1中可见。如图注所述，执行叶节点的原因可从其所属子树名称清晰得知。此内容将在第 8节关于可解释人工智能部分进一步讨论。

行为树的一个关键要素是图 2中操作区域 $\boldsymbol{\Omega}_i$ 如何依赖于层次结构中所有子树的成功、失败和运行区域 S_i, F_i, R_i。因此，我们需要确定这些集合的一些性质。

Lemma 1. *若 $\mathscr{T}_0 = Sequence(\mathscr{T}_1, \mathscr{T}_2)$，则定义 3 蕴含：*

$$S_0 = S_1 \cap S_2, \tag{5.}$$

$$F_0 = F_1 \cup (S_1 \cap F_2), \tag{6.}$$

$$R_0 = R_1 \cup (S_1 \cap R_2), \tag{7.}$$

Proof. 直接应用定义即可得到上述结论。 □

考虑图 3中的移动操作臂示例，实际上是图 1的子集，其中移除了子树 6 和 37，并将子树 3、11 和 34 合并为单节点。这样做是为了展示模块化如何使分析在不同层级上进行。根节点 0 是子树 1 和 9 的序列组合。方程式 (2) 说明“确保顶层目标达成（0）”仅在“确保处于安全区域（1）”返回成功 $x \in S_1$ 时，执行“确保物体到达目标（9）”，即 $u_0(x) = u_9(x)$。否则，执行节点 1，即 $u_0(x) = u_1(x)$。类似地，方程式 (6) 表示当节点 1 返回失败（无法到达安全区域），或者节点 1 返回成功且节点 9 返回失败（处于安全区域但无法将物体送达目标）时，节点 0 返回失败。

Definition 4. （行为树的回退组合） 两个或多个行为树可以用回退算子组合成更复杂的行为

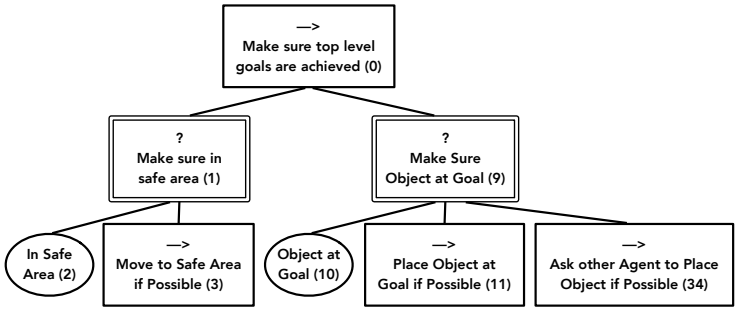


Figure 3: 该移动操作臂行为树为图 1的子集，整体目标是在工作区的安全区域内将指定物体移动到目标区域。所有节点均以对应子树的索引编号。序列节点 0, 3, 11, 34 用符号 (→) 表示，回退节点 1, 9 用符号 (?) 表示。条件节点以椭圆形标示。注意节点 3, 11, 34 的子节点未在此图中展示。

树，$\mathscr{T}_0 = \mathit{Fallback}(\mathscr{T}_1, \mathscr{T}_2)$. 则 r_0, u_0 定义如下：

$$\text{若} x \in F_1: \qquad r_0(x) = r_2(x),\ u_0(x) = u_2(x) \tag{8.}$$

$$\text{否则}: \qquad r_0(x) = r_1(x),\ u_0(x) = u_1(x) \tag{9.}$$

注意执行新的行为树 $\mathscr{T}_0$ 时，首先保持执行其第一个子节点 $\mathscr{T}_1$，只要其返回 运行中或 成功（$x \notin F_1$），即执行式 (9)。仅当第一个子节点返回 失败（$x \in F_1$）时，执行第二个子节点 $\mathscr{T}_2$，即执行式 (8)。最终，回退节点 $\mathscr{T}_0$ 仅当所有子节点均尝试失败时返回 失败（$x \in F_1 \cap F_2$），因此得名“回退”。

为方便表示，写作

$$\text{Fallback}(\mathscr{T}_1, \text{Fallback}(\mathscr{T}_2, \mathscr{T}_3)) = \text{Fallback}(\mathscr{T}_1, \mathscr{T}_2, \mathscr{T}_3), \tag{10.}$$

任意长度的回退组合类似。回退节点亦用符号（?）表示，如图 3所示。

Lemma 2. *若 $\mathscr{T}_0 = \mathit{Fallback}(\mathscr{T}_1, \mathscr{T}_2)$，则定义 4 蕴含：*

$$S_0 = S_1 \cup (F_1 \cap S_2), \tag{11.}$$

$$F_0 = F_1 \cap F_2, \tag{12.}$$

$$R_0 = R_1 \cup (F_1 \cap R_2), \tag{13.}$$

Proof. 直接应用定义即可得到上述结论。 □

Definition 5 (条件节点)**.** *若行为树 $\mathscr{T}_i$ 满足 $R_i = \emptyset$，则称其为条件节点。作为行为树，它仍定义有 u_i，但如引理 8 所示，该控制不会被执行。*

再次考虑图 3中的移动操作臂示例。“确保处于安全区域（1）”是节点 2 和 3 的回退组合。

方程式 (8) 表明节点 1 仅在“处于安全区域（2）”返回失败 $x \in F_2$ 时，执行“尽可能移动到安全区域（3）”，即 $u_1(x) = u_3(x)$。此外，节点 2 是条件节点，$R_2 = \emptyset$，故当 $x \notin F_2$ 时，有 $x \in S_2$，节点 1 返回成功，并将成功状态传递至节点 0，后者执行“确保物体到达目标（9）”，依此类推。进一步，方程式 (12) 表明节点 1 失败的唯一情况是节点 2 和 3 同时失败，即“确保处于安全区域（1）”仅当“处于安全区域（2）”和“尽可能移动到安全区域（3）”均失败时才返回失败。

至此，我们具备创建和执行行为树的所有必要条件。下一节将探讨行为树的模块化特性，并分析在何种情况下执行能收敛至成功区域。

4. 最优模块化

行为树的一个关键优势是其模块化，这一特性得益于所有层级子树均具有相同接口（见定义 1）。然而，如文献 (19) 所示，通过扩展图论中模块化/复杂度的度量，可进行更深入的分析。本节将简要介绍关键理论结果，证明行为树的环路复杂度（cyclomatic complexity）为一，这使得它们在某一类控制结构中具有*最优模块化*特性。

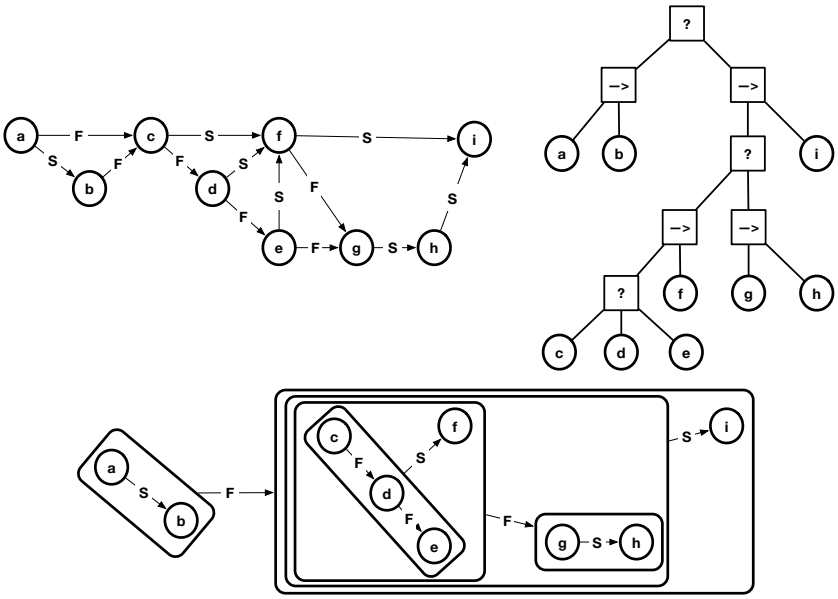


Figure 4: 行为树（右上）及其对应的不连续系统（左上），以及该不连续系统的模块分解（下方）。注意分解中的所有图均对应无环路径。根据定理 2，所有行为树对应的不连续系统都具有此类无环图，从而所有行为树的环路复杂度均为一。

为了研究一般反应式控制架构中的模块化概念，文献 (19) 中定义了所谓的决策结构（Decision Structures，DS）。这些是有向图，如图 4左上部分所示。结构中的每个节点对应一个控制器 u_i，每条边的标签对应该边出发节点可能返回的状态 r_i。决策结构的执行方式如下：从源节点开始（见图 4左上），查看该节点的返回状态 $r_i(x)$，沿着与 $r_i(x)$ 标签对应的边（如果存

在）继续。同样检查新节点的返回状态并沿相应的边前进，直到遇到某个 $r_i(x)$ 没有对应的出边标签为止。此时选择该节点的控制器 u_i。该过程不断从源节点迭代，持续确定应执行的控制器。

结合以上描述，图 4中左上方决策结构的执行与右上方行为树的执行完全相同。因此，该决策结构与行为树等价。实际上，决策结构是行为树的推广，所有行为树均可表示为决策结构，但并非所有决策结构都能表示为行为树。决策结构较为通用，标签集大小任意，但需有限。因此，决策结构与有限状态机（FSM）非常相似，区别在于决策结构不断从源节点开始，以确定应执行的控制器。

借鉴图论中模块的概念 (20)，文献 (19) 的作者对决策结构（DS）中的模块作出如下定义。

Definition 6 (文献 (19) 中定义 6.3，决策结构中的模块)**.** *设 Z 为一个决策结构。令 $Y \subset N(Z)$ 是节点的子集，且 $Z[Y]$ 也是一个决策结构。若对所有节点 $v \in N(Z) \setminus Y$，从 v 指向 Y 的弧均指向 Y 的源节点，且若存在标签为 r 的弧从 Y 指向 v，则对所有 $y \in Y$，标签为 r 的弧均存在且要么指向 v，要么指向 Y 内的其他节点，则称 Y 是一个模块。*

随后定义了商决策结构（quotient DS），即将模块折叠为单一节点。

Lemma 3 (文献 (19) 中引理 6.8)**.** *设 Z 为决策结构，P 为模块划分，则商图 Z/P 也是一个决策结构。若 P 为极大划分，则 Z/P 是素的。*

基于此，定义了极大模块分解（有兴趣者可参考文献 (19)），从而得到以下定理。

Theorem 1 (文献 (19) 中定理 6.15)**.** *设 Z 为具有 k 个不同弧标签的决策结构，则当且仅当 Z 的模块分解中所有商图均为路径时，Z 结构上等价于一个 k-BT。*

Proof. 详见文献 (19)。 □

这里，k-BT 是行为树的推广，标签集大小为 k。其中，2-BT 对应普通行为树，计入成功和失败标签，但不计运行标签（因其不导致决策结构中的转换）。k-BT 还有 k 种不同的内部节点，是对普通行为树中序列（Sequence）和回退（Fallback）两个节点的推广。

该定理在图 4中有清晰体现，我们可见决策结构的模块分解中无环。循环数被证明与代码测试和调试难度相关 (21)。因此，环路复杂度（cyclomatic complexity）被定义于图中，文献 (19) 的作者将其扩展至决策结构：

Definition 7 (文献 (19) 中定义 6.19)**.** *设 Z 为决策结构。Z 的环路复杂度定义为 Z 中线性无关的无向环的数量加一。*

随后定义扩展以考虑模块化：

Definition 8 (文献 (19) 中定义 6.21)**.** *设 Z 为决策结构。Z 的本质复杂度定义为其模块分解中所有商图的最大环路复杂度。*

最终证明以下定理：

Theorem 2 (文献 (19) 中定理 6.23)**.** *设 Z 为具有 k 个不同边标签的决策结构。Z 当且仅当本质复杂度为 1 时，等价于一个 k-BT。*

Proof. 详见文献 (19)。 □

针对 $k=2$（两个不同边标签，成功 S 和失败 F）的情况，定理说明行为树恰为本质复杂度为 1 的决策结构。因此，行为树对应一类最优模块化的决策结构。

5. 收敛性证明

许多控制问题的目标是使某平衡点稳定，即大量从不同状态出发的轨迹最终收敛至该平衡点。对于行为树，我们不选取具体平衡点，而假设根节点的成功区域 S_0 表示期望结果集合，设计目标即使大量状态轨迹收敛至 S_0 内的点。因此，本节研究在何种条件下可保证状态最终进入 S_0。

主要结果是行为树的一般收敛性证明——定理 3，并附若干示例。我们将尝试利用行为树的模块化特性，使结果适用于各层抽象，既可将整个子树视为单一实体（如图 3），也可分解为组成部分（如图 1）。

证明思路直观，如图 5所示。与图 2类似，状态空间被划分为操作区域 Ω_i、整体失败区域 F_0 和成功区域 S_0，箭头表示这些集合间可能的状态转移。若某操作区域 Ω_i 的所有转移均指向 S_0 或某 Ω_j（且 $j>i$），且状态不会无限停留在 Ω_i，则最终状态必然达到 S_0。注意，该分析可在多个抽象层级上进行。例如若 $\Omega_6=\Omega_4\cup\Omega_5$，可分别考虑 Ω_4,Ω_5（如图 5(a)），也可合并考虑（如图 5(b)）。

(a) (b)

Figure 5: 定理 3背后的思想。若状态只在由箭头连接的集合间转移，且不会无限停留在任意 Ω_i 中，最终将到达 S_0。

5.1. 一般结果

集合 Ω_i 定义如下，代表控制器 u_i 正在运行的区域，详见引理 8。但要定义 Ω_i，需先定义影响区域 I_i，即状态空间中 $\mathcal{S}_i$ 的变化可能影响 u_0 的部分。为定义 I_i，需先定义节点的父节点

和“哥哥节点”(左侧最近的兄弟节点)。以下部分结果采自文献 (13, 14, 15)。

Definition 9 (节点的父节点和哥哥节点)**.** *给定节点 i，令 $p(i)$ 为该节点的父节点，$b(i)$ 为该节点左侧最近的兄弟节点（哥哥节点）。*

注意，若 i 是根节点，则 $p(i)$ 未定义；若无左侧兄弟，则 $b(i)$ 未定义。

Definition 10 (影响区域)**.** *节点 i 的影响区域 I_i 定义为：*

$$I_i = X \qquad \text{如果} i \text{是根节点} \tag{14.}$$

$$I_i = I_{p(i)} \qquad \text{如果} i \text{是最左侧的兄弟节点且} \exists p(i) \tag{15.}$$

$$I_i = I_{b(i)} \cap S_{b(i)} \qquad \text{如果} p(i) \text{是序列节点且} \exists b(i) \tag{16.}$$

$$I_i = I_{b(i)} \cap F_{b(i)} \qquad \text{如果} p(i) \text{是回退节点且} \exists b(i) \tag{17.}$$

注意，影响区域 I_i 是状态空间中设计 $\mathscr{T}_i = (u_i, r_i)$ 会影响 $\mathscr{T}_0$ 执行的部分。且 I_i 与 S_i, F_i, R_i 本质不同，I_i 完全依赖于 $\mathscr{T}_0$ 中 $\mathscr{T}_i$ 以外的部分，即父节点和兄弟节点，而 S_i, F_i, R_i 则完全依赖于 $\mathscr{T}_i$ 内部，即 $u_i(x)$。

Lemma 4. *若 $x \notin I_i$，则改变 u_i, r_i 的实现不会改变 $u_0(x)$ 的值。*

Proof. 利用下述引理 8，若 $x \in \Omega_i = I_i \cap R_i$，则 $u_0(x) = u_i(x)$。为最大化 u_i, r_i 的影响，将 $r_i(x)$ 设为总是返回运行状态，使得 $R_i = X$，$\Omega_i = I_i$。但若 $x \notin I_i$，则 $x \notin \Omega_i$，执行仍由其他子树控制。 □

如上所述，I_i 取决于外部因素，S_i, F_i, R_i 取决于内部因素。现在定义 Ω_i 为 $u_0(x) \equiv u_i(x)$ 且 $r_i(x) = R$ 的区域，即 $\mathscr{T}_i$ 控制执行的区域。

Definition 11 (操作区域)**.** *节点 i 的操作区域 Ω_i 定义为：*

$$\Omega_i = I_i \cap R_i \tag{18.}$$

Lemma 5. *对于给定节点 j，其子节点的操作区域构成 Ω_j 的划分，即：*

$$\bigcup_{i:p(i)=j} \Omega_i = \Omega_j, \tag{19.}$$

$$\Omega_i \cap \Omega_k = \emptyset, \quad \forall i,k : i \neq k, p(i) = p(k) \tag{20.}$$

Proof. 设父节点索引为 0，两个子节点索引为 1 和 2。需证明此结论对序列和回退组合均成立。

若父节点为序列，则 $R_0 = R_1 \cup (S_1 \cap R_2)$。假设给定 I_0，则 $I_1 = I_0$，$I_2 = I_1 \cap S_1 = I_0 \cap S_1$。

则有:

$$\Omega_0 = I_0 \cap R_0 = I_0 \cap (R_1 \cup (S_1 \cap R_2)) \tag{21.}$$

$$\Omega_1 = I_1 \cap R_1 = I_0 \cap R_1 \tag{22.}$$

$$\Omega_2 = I_2 \cap R_2 = I_0 \cap S_1 \cap R_2 \tag{23.}$$

故 $\Omega_1 \cap \Omega_2 = I_0 \cap R_1 \cap I_0 \cap S_1 \cap R_2 = \emptyset$，因 $R_1 \cap S_1 = \emptyset$。且

$$\Omega_1 \cup \Omega_2 = (I_0 \cap R_1) \cup (I_0 \cap S_1 \cap R_2) = I_0 \cap (R_1 \cup (S_1 \cap R_2)) = I_0 \cap R_0 = \Omega_0.$$

类似地,若父节点为回退,则 $R_0 = R_1 \cup (F_1 \cap R_2)$。假设给定 I_0,则 $I_1 = I_0, I_2 = I_1 \cap F_1 = I_0 \cap F_1$。则有:

$$\Omega_0 = I_0 \cap R_0 = I_0 \cap (R_1 \cup (F_1 \cap R_2)) \tag{24.}$$

$$\Omega_1 = I_1 \cap R_1 = I_0 \cap R_1 \tag{25.}$$

$$\Omega_2 = I_2 \cap R_2 = I_0 \cap F_1 \cap R_2 \tag{26.}$$

故 $\Omega_1 \cap \Omega_2 = I_0 \cap R_1 \cap I_0 \cap F_1 \cap R_2 = \emptyset$，因 $R_1 \cap F_1 = \emptyset$。且

$$\Omega_1 \cup \Omega_2 = (I_0 \cap R_1) \cup (I_0 \cap F_1 \cap R_2) = I_0 \cap (R_1 \cup (F_1 \cap R_2)) = I_0 \cap R_0 = \Omega_0.$$

□

Lemma 6. 对于给定子树，其叶节点的操作区域构成根节点操作区域的划分。

Proof. 递归应用引理 5。 □

如上所述，期望在不同抽象层级上进行收敛分析，如图 5所示。因此定义如下。

Definition 12. 抽象层级 $L \subset \{0,1,2,\ldots\}$ 是索引集合，满足:

$$X = \bigcup_{i \in L} \Omega_i \cup S_0 \cup F_0,$$

且 $\Omega_i \cap \Omega_j = \emptyset, \forall i \neq j \in L$。

Lemma 7. 根节点自身构成一个抽象层级，即 $L = \{0\}$，所有叶节点构成另一个抽象层级，即 $L = \{$叶节点索引$\}$。

Proof. 根节点有 $\Omega_0 = I_0 \cap R_0 = X \cap R_0 = R_0$，因此

$$\bigcup_{i \in L} \Omega_i \cup S_0 \cup F_0 = R_0 \cup S_0 \cup F_0 = X.$$

若根节点成立，则叶节点亦成立，因为根据引理 6，

$$\bigcup_{i\in L}\Omega_i = \Omega_0.$$

□

Lemma 8. *若 $\mathcal{T}_i$ 是 $\mathcal{T}_0$ 的子树，则 $x\in\Omega_i$ 蕴含 $u_0(x)=u_i(x)$，即当状态位于 Ω_i 时，控制器 i 被执行。*

Proof. 根节点成立，因为 $\Omega_0 = I_0\cap R_0 = R_0$。假设父节点 $j=p(i)$ 成立，现证明子节点也成立，即需证 $u_j(x)=u_i(x)$。已知 $x\in\Omega_i\subset\Omega_j$，且 $\Omega_i = I_i\cap R_i$。

若 j 是最左子节点，则 $x\in R_i$ 蕴含 $x\notin S_i\cup F_i$，根据式 (2) 和 (8) 有 $u_j(x)=u_i(x)$。

假设父节点为序列节点，若 j 不是最左子节点，则 $x\in I_i$ 蕴含 $x\in S_{b(i)}$，根据式 (3) 和 (16) 有 $u_j(x)=u_i(x)$。

反之，假设父节点为回退节点，若 j 不是最左子节点，则 $x\in I_i$ 蕴含 $x\in F_{b(i)}$，根据式 (9) 和 (17) 有 $u_j(x)=u_i(x)$。 □

基于上述概念，我们现在可以阐述关于行为树收敛性的主要定理，灵感来源于文献 (22, 23, 24)。其思想是：若状态沿操作区域 Ω_i 严格递增顺序移动，且在任一区域停留不超过 τ 时间，且唯一允许的其他区域为成功区域 S_0，则系统将在有限时间内达到 S_0。形式化表述如下：

Theorem 3 (行为树的收敛性)**.** *给定行为树，一个需保持不变的外部约束区域 $\bar{C}\subset X$，以及一个抽象层级 L。若存在对 L 中的 N 个节点的重标号，使得*

$$C_i = \Big(\big(\bigcup_{j\in L, j\geq i}\Omega_j\big)\cup S_0\Big)\cap\bar{C} \tag{27}$$

对所有 $i\in L$ 在控制器 u_i 下保持不变，且存在 $\tau>0$ 使得若 $x(t)\in\Omega_i$，则 $x(t+\tau)\notin\Omega_i$，则存在时间 $t'\leq N\tau$，使得若初始状态 $x(0)\in C_1$，则 $x(t')\in S_0$。

Proof. 根据式 (27)，有 $C_1\supseteq C_2\supseteq\cdots\supseteq C_N$。若 $x(t)\in\Omega_i$，则 $x(t+\tau)\notin\Omega_i$，且 C_i 对 u_i 不变，故 $x(t+\tau)\in\Omega_j$ 且 $j>i$ 或 $x(t+\tau)\in S_0$。因此，最多经历 N 次区域跳转后，状态将进入 S_0，所需时间不超过 $N\tau$。 □

当称集合 $B\subset X$ 在 u_i 下不变时，意指若初始状态 $x(0)\in B$ 且动力学为 $\dot{x}=f(x,u_i(x))$，则 $x(t)\in B$ 对所有 $t>0$ 均成立。离散时间情形亦类似，$x_{t+1}=f(x_t,u_i(x))$。

Remark 3. *注意，证明行为树收敛性的挑战在于选择合适的抽象层级、对节点 L 重排序，以及设计 u_i 使得 C_i 保持不变且满足 $x(t+\tau)\notin\Omega_i$。*

Remark 4. *外部约束区域 $\bar{C}$ 的目的是允许对用作另一行为树子树的行为树进行单独分析。若不需要，可设 $\bar{C}=X$。*

上述确定性分析可由文献 (25) 中的概率性结果补充。

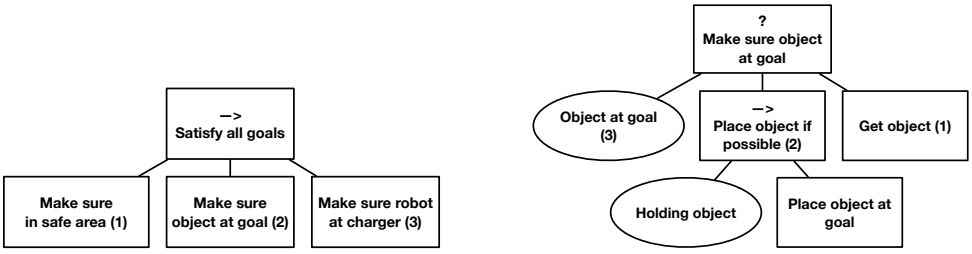
Lemma 9 (概率性转移). *若定义 2中的执行被非确定性转移替代，且控制器 u_i 满足从 Ω_i 转移到索引更小的 Ω_j（$j<i$）的非期望转移概率为 $1-p^i$，其中 $0<p\leq p^i<1$，则期望转移次数 T 在达到成功区域 S_0 之前有上界 $E(T)\leq \frac{N}{p^N}$，且最多经过 k 次转移达到目标的概率为 $P_k=1-\gamma^{k+1}$，其中 $\gamma=1-p^N$，使得 $P_\infty=1$。*

Proof. 详见文献 (25)。 □

上述引理有两种解释：一种是针对非确定性执行，如引理所述；另一种是针对确定性执行，其中未建模的外部代理引发状态有限跳转。同时，期望转移次数给出了期望收敛时间的上界 $\tau N/p^N$。

5.2. 三个示例

现将定理 3 应用于：图 6(a) 中的目标序列，图 6(b) 中的回退动作序列，每个动作旨在满足其左侧动作的前置条件，以及更复杂的移动操作臂行为树（图 1）。结果集合展示于表 1。如表所示，需保持不变的集合 C_i 往往不复杂，设计满足其的控制器 u_i 也通常合理。



(a) (b)

Figure 6: 定理 3 的两个应用示例。节点的索引标签对于序列从左到右，对于回退则从右到左。此为应用定理时重标号的结果。

Lemma 10 (标准序列). *若 $\mathscr{T}_0=Sequence(\mathscr{T}_1,\ldots,\mathscr{T}_N)$，则有*

$$I_i=\bigcap_{j<i} S_j\cap I_{p(i)} \tag{28.}$$

$$\Omega_i=\bigcap_{j<i} S_j\cap I_{p(i)}\cap R_i \tag{29.}$$

$$C_i=\left(\bigcap_{j<i} S_j\cup S_0\right)\cap \bar{C} \tag{30.}$$

Proof. 由序列节点定义，I_i, Ω_i 明确仅在前序子节点成功时执行下一个子节点。由定理 3，有

$$C_i = \left(\bigcup_{j \geq i} \Omega_j \cup S_0 \right) \cap \bar{C}.$$

故需证明

$$\bigcup_{j \geq i} \Omega_j = \bigcap_{j < i} S_j.$$

假设 $x \in \bigcap_{j<i} S_j$，则 $x \in S_j$ 对所有 $j < i$ 成立，故 $x \notin \Omega_j$ 对所有 $j < i$ 成立。由于 Ω_i 构成 $\Omega_{p(i)}$ 的划分，存在 $k \geq i$ 使得 $x \in \Omega_k$。反之，若 $x \in \bigcup_{j \geq i} \Omega_j$，则存在 $k \geq i$ 使得 $x \in \Omega_k$，从而 $x \in \bigcap_{j<i} S_j$。 □

在下一个示例中，我们采用设计原则隐式序列（implicit sequence）(12, 25)，其中回退节点的每个子节点被设计为满足其左侧兄弟节点的前置条件，且不失败，见下式 (33) 中的 $\hat{C}_i \subset C_i$。因此，编号也作为应用定理 3的一部分，从右向左进行。

Lemma 11 (隐式序列)**.** *若 $\mathscr{T}_0 = Fallback(\mathscr{T}_N, \ldots, \mathscr{T}_1)$，且对所有 $i < N$ 存在 $j > i$ 满足 $S_j \cup R_j \supset S_i$，则有*

$$I_i = \bigcap_{j>i} F_j \cap I_{p(i)} \tag{31.}$$

$$\Omega_i = \bigcap_{j>i} F_j \cap I_{p(i)} \cap R_i \tag{32.}$$

$$C_i \supset \hat{C}_i = (R_i \cup S_i) \cap \bar{C} \tag{33.}$$

Proof. I_i, Ω_i 根据回退节点定义显然，只有前一个子节点失败时才执行下一个子节点。此外，$S_0 = S_N$，因为对于 $i \neq N$，有 $S_i \cap I_i = \emptyset$，这是由于存在某个 $j > i$ 使得 $S_j \cup R_j \supset S_i$。

由定理 3，有

$$C_i = \left(\bigcup_{j \geq i} \Omega_j \cup S_0 \right) \cap \bar{C}.$$

故需证明

$$\bigcup_{j \geq i} \Omega_j \cup S_N \supset R_i \cup S_i \cup S_N.$$

假设 $x \in R_i \cup S_i \cup S_N$，则 $x \notin F_i$，故 $x \notin \bigcup_{j<i} \Omega_j$，因此

$$x \in \bigcup_{j \geq i} \Omega_j.$$

□

利用上述结果，可计算各操作区域 Ω_i 对应需由控制器 u_i 保持不变的集合 C_i。进而可构

Table 1: 应用定理 3的三个示例。

名称	目标	操作区域 Ω_i	需保持不变的集合 C_i
图 6(a)			
$\mathscr{T}_1$: 确保处于安全区域	处于安全区域	非处于安全区域	空集 $\emptyset$
$\mathscr{T}_2$: 确保物体到达目标	物体到达目标	处于安全区域且非物体到达目标	处于安全区域
$\mathscr{T}_3$: 确保机器人到达充电器	机器人到达充电器	处于安全区域且物体到达目标且非机器人到达充电器	处于安全区域且物体到达目标
图 6(b)			
$\mathscr{T}_1$: 获取物体	抓握物体	非抓握物体且非物体到达目标	空集 $\emptyset$
$\mathscr{T}_2$: 可行则放置物体	物体到达目标	抓握物体且非物体到达目标	抓握物体
图 1			
$\mathscr{T}_5$: 移动到安全区域	处于安全区域	非处于安全区域	空集 $\emptyset$
$\mathscr{T}_8$: 充电	电池电量充足	处于安全区域且非电池电量充足	处于安全区域
$\mathscr{T}_{19}$: 移动到物体	机器人靠近物体	处于安全区域且电池电量充足且非物体到达目标且非物体夹持且非机器人靠近物体且存在通向物体的通路	处于安全区域且电池电量充足
$\mathscr{T}_{20}$: 左臂抓取物体	物体夹持	处于安全区域且电池电量充足且非物体到达目标且非物体夹持且机器人靠近物体	处于安全区域且电池电量充足
$\mathscr{T}_{21}$: (因篇幅略去)			
$\mathscr{T}_{32}$: 移动到目标	机器人靠近目标	处于安全区域且电池电量充足且非物体到达目标且物体夹持	处于安全区域且电池电量充足且物体夹持
$\mathscr{T}_{33}$: 放置物体到目标	物体到达目标	处于安全区域且电池电量充足且非物体到达目标	处于安全区域且电池电量充足
$\mathscr{T}_{36}$: 请求其他代理放置物体	物体到达目标	处于安全区域且电池电量充足且非物体到达目标且子树 $\mathscr{T}_{11}$ 返回失败	处于安全区域且电池电量充足且子树 $\mathscr{T}_{11}$ 返回失败
$\mathscr{T}_{39}$: 移动到充电器	机器人在充电器	处于安全区域且电池电量充足且物体到达目标	处于安全区域且电池电量充足且物体到达目标

建类似表 1的表格，用作各 u_i 的设计规范。

表 1 还包含图 1中更复杂行为树的结果，节点编号由树的深度优先遍历决定。接下来我们将探讨如何根据动作列表及对应的前置条件和后置条件，递归地创建此类行为树。

6. 利用模块化与反馈的设计原则

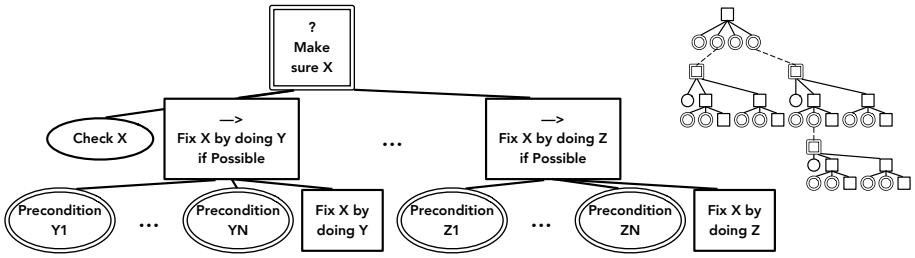


Figure 7: 左侧展示了一个基础行为树，用于通过执行动作 Y 或动作 Z 来实现某个条件 X。注意图 1 中的行为树是通过连接八个此类形式的行为树构建的。右侧示意了从包含四个最高优先级目标的序列节点开始，递归地用实现相应条件的行为树根节点替换叶节点条件（虚线所示）。图中及图 1中，可被替换的条件和已被回退节点替换的条件均以双线框标示。

本节将具体展示行为树中层次模块化与反馈的应用。考虑图 7中的示例行为树，设计目标是确保某条件 X 成立。若 X 已成立，则立即返回成功；否则将尝试使 X 成立，因而得名“确保 X”。有时使 X 成立的方式不止一种，这些备选方案被收集于回退节点下，若某一选项（如 Y）失败或不适用，则调用另一选项（如 Z）。Y 和 Z 各自有前置条件，描述何时可调用，如图所示。

关键思想是递归应用图中左侧的设计，如右侧所示。首先，将四个最高优先级目标放入序列节点。然后，非仅检查条件，而是用左侧形式的小行为树替换条件，尝试使其成立。结果行为树会产生新条件，这些条件又可继续被替换为小行为树，依此递归。图 1和图 7中，所有可替换或已替换的条件均以双线框标示。注意，不应替换单线框的条件，如“检查 X”，因其已有实现动作。

此递归方法充分体现了行为树支持的层次模块化。图 7的设计仅关注实现 X，而非为何实现或后续影响。同时体现反馈机制：行为树首先检测是否需要执行操作，若不需则返回成功；若需，则尝试实现，若某选项失败，则尝试备选方案。

文献 (14) 对上述设计进行了详尽分析，讨论了其适用性与局限性。这里指出，应用定理 3可得表 1中的集合。执行顺序应遵循索引递增顺序，故表格可自上而下阅读。注意集合 C_i 多对应不违反先前达成的子目标，如“处于安全集”和“电池电量充足”；但在将物体搬至目标时，还包括“物体夹持”条件。或许最令人意外的是 $\mathcal{T}_{36}$ 的 C_{36}，包含“子树 $\mathcal{T}_{11}$ 返回失败”。此条件避免出现某选项初次失败后，回退选项执行时该选项又被激活且再次失败，导致在选项间反复切换的情况。详见文献 (14)。

7. 利用控制屏障函数保证安全性与不变性

本节将探讨如何利用控制屏障函数（Control Barrier Functions，CBFs）在某些情况下提供定理 3 所需的不变性保证。此外，将展示使用 CBF 保证安全性的标准方法如何作为定理 3 的特例出现。最后讨论如何处理目标冲突问题，因为同时保持多个集合不变并非总是可行。

7.1. 控制屏障函数

如上文第 5 节所述，定理 3 的关键是控制器 u_i 保持集合 C_i 不变。而保持集合不变正是 CBF 的主要目标。以下假设定义 2 中的执行为连续时间，离散时间概念亦可对应构建。

CBF 的核心思想 (26, 27) 是定义屏障函数 $h : X \to \mathbb{R}$，使所谓的*安全集* $\mathscr{C}$ 被刻画为：

$$\mathscr{C} = \{x \in X : h(x) \geq 0\}.$$

给定连续系统动力学 $\dot{x} = f_C(x,u)$，若选取控制 $u \in U_{inv}$，其中

$$U_{inv} = \{u \in U : \frac{dh}{dx} f_C(x,u) \geq -\alpha(h(x))\}, \tag{34.}$$

且 $\alpha \in \mathscr{K}$ 是 K 类函数 (26)，则系统保证保持在安全集内，即 $x \in \mathscr{C}$。

7.2. 保证安全性与处理目标冲突

安全性通常是最高优先级目标。本节将展示如何利用 CBF 处理定理 3 的不变性要求，使安全保证成为其特例，如文献 (28) 所述。

由于需保持条件集 C_i 不变，首先做如下假设：

Assumption 2. *每个条件 $C_i : X \to \{0,1\}$ 可通过 CBF h_i 表述，见式 (34)，具体为：*

$$C_i = \{x \in X : h_i(x) \geq 0\}.$$

单一 CBF 可保证不变性，若存在多个，可能导致目标冲突，例如“处于安全区域”与“位于充电器处”若充电器不在安全集内。通常，相关控制集合 U_{inv} 的交集可保证所有集合不变，但若目标冲突，交集可能为空。此时，利用行为树中目标的优先级顺序（如“处于安全区域”优先于“位于充电器”），策略是尽可能多地将集合包含于交集中，同时确保交集非空。

据此定义以下控制集，其中 $U_i \subset U$ 保证 C_i 不变，$\bar{U}_i \subset U$ 保证所有 $C_j, j \leq i$ 不变（可能为空），$\hat{U}_i \subset U$ 保证部分 $C_j, j \leq i$ 不变（且保证非空）。

Definition 13. *设*

$$U_i = \{u \in U : \frac{dh_i}{dx} f(x,u) \geq -\alpha(h_i(x))\} \tag{35.}$$

$$\bar{U}_i = \bigcap_{j=1}^{i} U_j \tag{36.}$$

$$\hat{U}_i = \bar{U}_j : j \leq i, \bar{U}_j \neq \emptyset \wedge (j = i \vee \bar{U}_{j+1} = \emptyset) \tag{37.}$$

控制器可在 $\hat{U}_i$ 内选取，使其尽可能接近设计用于实现当前子目标的期望控制 $w_i(x)$，如文

献 (26) 中的 CBF-QP：

$$u_i = \text{argmin}_u ||u - w_i(x)||^2 \quad \text{s.t. } u \in \hat{U}_i \tag{38.}$$

若将此方法应用于任意复杂行为树（如图 1），且将安全目标设为第一优先级，则上述 CBF 方法能保证机器人永远不违反该目标。理想情况下，可实现所有目标，但机器人安全性始终得以保障。

8. 可解释 AI 及人机交互

随着机器人与人类共用工作空间的程度日益加深，人机交互问题愈发重要。如上所述，安全性通常是首要考虑，但为实现高效交互，人类需能预测、信任并理解机器人行为。

文献 (29) 提出了可信自主系统若干指南，其中包括系统应具备透明性和可追踪性，具体表现为“系统必须能够以简洁且可用的形式（视觉或文本）明确解释其推理过程”。如图 1所示，行为树满足此要求，因为任何时刻均可定位执行中的叶节点，并沿分支追溯至根节点，了解为何该子树被执行。若采用第 6节中描述的递归向后链式方法，阅读展开的前置条件（图中以双线框标示）即可看到机器人当前执行“移动到物体”，以实现“确保机器人靠近物体”，进一步为“确保物体夹持”，最终为“确保物体到达目标”。行为树在人机交互方面的用户研究尚需加强，早期相关工作见文献 (30)。

9. 强化学习、效用与行为树

强化学习（Reinforcement Learning, RL）旨在为广泛的控制问题家族设计近似最优控制器。有时可以找到所谓的端到端解决方案，直接将原始传感器数据映射为动作，解决极具挑战性的问题 (31)。如果所有问题都能用 RL 端到端解决，则无需行为树（BT），但由于模块化层次结构支持安全保证（见第 7节）及对人类操作员的透明性（见第 8节），故预计模块化层次控制结构在未来仍将保持重要。一个自然的问题是如何将 RL 与 BT 结合，理想地既能获得 RL 的性能，又能享受 BT 的保证与透明性。

图 8展示了多种结合 RL 与 BT 的方法。最直观的是用 RL 替换单个动作（图中 1），此方案在文献 (32, 33) 中有探讨，用户需指定状态、动作及奖励。若问题领域适合 RL，可逐步拓展，自底向上替换子树（图中 2），该渐进方式风险较低，便于替代整个控制结构。

RL 亦可用于提升现有行为树性能。一种方法是保留现有结构，新增 RL 选项，如文献 (34) 所示（图中 3）。若 RL 选项失败，其他选项仍将执行以完成子目标。

另一方法聚焦于子树顺序调整（图中 4），见文献 (35, 36, 37, 38)。因 RL 中状态-动作对的 Q 值估计未来奖励，故可用 Q 值决定回退选项顺序，基于当前状态重新排序。文献 (39) 采用类似思路，通过执行时收集数据估计各子节点成功概率，并动态调整顺序，优先执行价值最

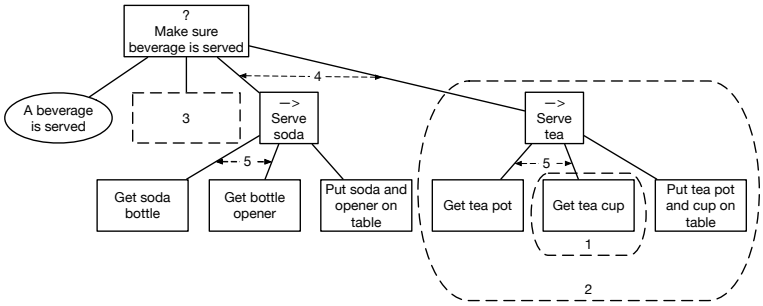


Figure 8: 结合 RL 与 BT 的多种方式。可以替换单个动作（1），或整个子树（2）；也可新增子树（3），保留旧子树作为回退选项；最后可调整回退选项顺序（4）或调整满足前置条件的顺序（5）。

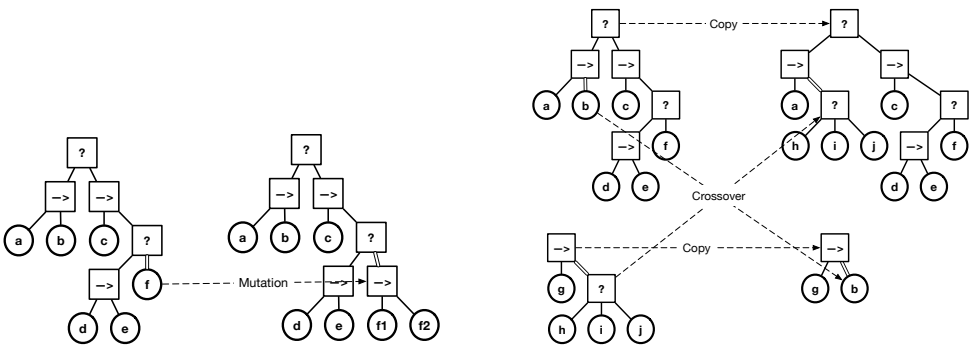
高者。

最后，较少探索的方式（图中 5）是重新排序行为树中的前置条件。例如，取回多个物品（图中所示）相当于旅行商问题（TSP）的小实例，顺序可能影响性能。

10. 进化算法与行为树

进化算法（或遗传算法）(40) 是受进化理论启发的局部优化算法。其基本思想是维护一组候选解，然后通过变异（对候选解的小幅修改）和交叉（选取两个候选解，标记各自子集并交换）产生新候选解。接着，利用适应度函数评估这些候选解，并淘汰部分个体。

行为树的模块化特性以及所有子树层级统一接口，使其非常适合应用进化算法。对行为树应用变异可通过选取任意子树并用其他子树替换来实现，如图 9(a) 所示。同理，交叉操作可通过选取两个行为树，分别选取子树并交换，如图 9(b) 所示。



(a) (b)

Figure 9: 对行为树应用变异和交叉操作

研究表明，设计的小幅变动引起性能的小幅变动的局部性，对于进化算法性能至关重要

(41)。如上文引理 4所示，影响区域 I_i 捕获了子树能影响系统行为的状态空间部分。因此，对于较大的行为树，特定子树的 I_i 往往较有限，体现了文献 (41) 中描述的局部性。

进化算法的一个著名问题是所谓的膨胀（bloating）(42)，即种群个体平均规模增长却不伴随适应度提升。对于行为树，设计中明显可能存在大面积无贡献部分。文献 (43) 和 (44) 均提出了解决该问题的方法。一种实用做法是尝试修剪不同子树，检测适应度是否下降；更理论的做法是计算所有子树的影响区域 I_i，移除那些 $I_i = \emptyset$ 的子树。

Work on BTs and evolutionary algorithms can be found in (45, 46, 43, 47, 48). Finally, combinations of BTs, evolutionary algorithms and planning can be found in (49), and multi-agent problems using BTs and evolutionary algorithms have been addressed in (50, 51).

11. 规划算法与行为树

规划算法通常用于生成一系列动作，将世界状态从给定的起始状态推进到期望的目标状态。规划可分为低层规划，如运动规划或抓取规划，或高层规划，如任务规划。低层规划通常作为行为树的叶节点集成，而高层规划则可用于构建整个行为树。结合行为树与规划算法的原因，通常是希望为规划器输出的目标导向动作添加行为树的反应式反馈特性。

使用任务规划器最直接的方式是先运行规划器获得动作序列，再执行该序列。当环境静态且动作可预测时，此方法效果良好。但若动作失败、感知不准确，或外部代理改变了世界状态，预定动作序列将无法达成目标。一种自然的反馈方式是监控执行状态，若出现显著偏差则重新规划。然而，许多情况下原规划依然有效，只需跳转至正确阶段即可。例如抓取失败时可重试；物体被拿起后又掉落时可回到抓取动作；外部代理协助完成子目标时可跳过至相关动作。正如前文所述，行为树是实现此类反馈控制的合适工具，如文献 (25) 所示。

规划还可用于生成反馈策略。一个例子是 A* 算法，它计算从所有初始状态（不仅是当前状态）到目标的最短路径。因此，若发生意外动作使状态偏离预期路径甚至移至非预期状态，规划依然包含正确动作。第 6节描述的设计在行为树中实现了该功能 (14)。其优点是吸引域较大，缺点是需覆盖所有潜在情形的行为树可能非常庞大。

行为树结合规划的另一应用见文献 (52)。该工作基于规划器输出创建行为树，意图通过反应式利用并行执行机会及任务提前完成，通过追踪每个动作的前置条件，实现尽可能早地执行动作，利用规划时不可得的信息。

行为树与规划器结合的研究可见于 (53, 54, 55, 56, 57, 58, 59, 49)。此外，高层任务网络（HTN）规划器的特例研究见于 (60, 61, 62, 63)，线性时序逻辑（LTL）规划器见于 (64, 65, 66)。

12. 结论

本文提供了基于控制理论的行为树（BT）方法，展示了行为树如何作为一种层次模块化结构，构建切换动力系统，其中切换基于来自低层模块的反馈。我们还展示了如何基于节点的父节点、兄弟节点和子节点的规格计算相应的操作区域。利用这些操作区域，提出了整体行

为树目标区域收敛的充分条件，并给出了可用于构建收敛行为树的实用设计方案。最后，阐明了这些核心成果与行为树相关的其他研究领域的联系，包括控制屏障函数、可解释人工智能、强化学习、遗传算法及规划。

要点总结

1. 行为树代表了一种层次模块化的方式，将多个控制器组合成更复杂的控制器。
2. 行为树支持各层级的反馈控制，接口中显式包含关于控制器适用性和执行进展的元信息（反馈），使父层级能够基于反馈采取行动。
3. 行为树的模块化结构便于对其收敛性和吸引域进行形式化分析。
4. 当前研究正致力于将行为树与规划、学习等其他领域连接起来。

未来研究问题

1. 强化学习能够端到端解决许多问题，但许多机器人系统仍需结合路径规划、抓取等独立能力的模块化结构。行为树是该结构的可行选择，强化学习与行为树的结合仍需深入探索。
2. 可解释人工智能、示范学习及人机交互是行为树透明性可能发挥重要作用的领域。
3. 虽然行为树已从人工智能和机器人视角被广泛研究，但从控制理论角度的研究仍然较少。

DISCLOSURE STATEMENT

The authors are not aware of any affiliations, memberships, funding, or financial holdings that might be perceived as affecting the objectivity of this review.

ACKNOWLEDGMENTS

The authors gratefully acknowledge the support from SSF through the Swedish Maritime Robotics Centre (SMaRC) (IRC15-0046), and by FOI through project 7135.

LITERATURE CITED

1. Blume M, Appel AW. 1999. Hierarchical modularity. *ACM Transactions on Programming Languages and Systems* 21(4):813–847
2. Sacerdoti ED. 1975. A Structure for Plans and Behavior. Tech. rep., SRI International AI center
3. Erol K, Hendler J, Nau DS. 1994. *UMCP: a sound and complete procedure for hierarchical task-network*

planning. In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems*, AIPS'94, pp. 249–254. Chicago, Illinois: AAAI Press

4. Mateas M, Stern A. 2002. A behavior language for story-based believable agents. *IEEE Intelligent Systems* 17(4):39–47
5. Isla D. 2005. *Handling Complexity in the Halo 2 AI*. In *Proceedings of the Game Developers Conference (GDC)*
6. Florez-Puga G, Gomez-Martin MA, Gomez-Martin PP, Diaz-Agudo B, Gonzalez-Calero PA. 2009. Query-Enabled Behavior Trees. *IEEE Transactions on Computational Intelligence and AI in Games* 1(4):298–308
7. Ögren P. 2012. *Increasing Modularity of UAV Control Systems using Computer Game Behavior Trees*. In *AIAA Guidance, Navigation, and Control Conference*. Minneapolis, Minnesota: American Institute of Aeronautics and Astronautics
8. Bagnell JA, Cavalcanti F, Cui L, Galluzzo T, Hebert M, et al. 2012. *An Integrated System for Autonomous Robotics Manipulation*. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2955–2962
9. Harel D. 1987. Statecharts: a visual formalism for complex systems. *Science of Computer Programming* 8(3):231–274
10. Biggar O, Zamani M, Shames I. 2021. An Expressiveness Hierarchy of Behavior Trees and Related Architectures. *IEEE Robotics and Automation Letters* 6(3):5397–5404
11. Iovino M, Scukins E, Styrud J, Ögren P, Smith C. 2020. A Survey of Behavior Trees in Robotics and AI. *arXiv:2005.05842 [cs]*
12. Colledanchise M, Ögren P. 2018. *Behavior Trees in Robotics and AI : An Introduction*. CRC Press
13. Colledanchise M, Ögren P. 2017. How Behavior Trees Modularize Hybrid Control Systems and Generalize Sequential Behavior Compositions, the Subsumption Architecture, and Decision Trees. *IEEE Transactions on Robotics* 33(2):372–389
14. Ögren P. 2020. Convergence Analysis of Hybrid Control Systems in the Form of Backward Chained Behavior Trees. *IEEE Robotics and Automation Letters* 5(4):6073–6080
15. Sprague CI, Ögren P. 2021. Continuous-time behavior trees as discontinuous dynamical systems. *IEEE Control Systems Letters* 6:1891–1896
16. Colledanchise M, Natale L. 2021. On the Implementation of Behavior Trees in Robotics. *IEEE Robotics and Automation Letters* :8
17. Cortes J. 2008. Discontinuous dynamical systems. *IEEE Control Systems Magazine* 28(3):36–73
18. Filippov AF. 1988. *Differential Equations with Discontinuous Righthand Sides: Control Systems*. Springer Science & Business Media
19. Biggar O, Zamani M, Shames I. 2020. On modularity in reactive control architectures, with an application to formal verification. *arXiv preprint arXiv:2008.12515*
20. Gallai T. 1967. Transitiv orientierbare Graphen. *Acta Mathematica Academiae Scientiarum Hungarica* 18(1):25–66
21. Watson AH, Wallace DR, McCabe TJ. 1996. *Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric*. U.S. Department of Commerce, Technology Administration, National Institute of Standards and Technology

22. Burridge RR, Rizzi AA, Koditschek DE. 1999. Sequential Composition of Dynamically Dexterous Robot Behaviors. *The International Journal of Robotics Research* 18(6):534–555
23. Conner DC, Choset H, Rizzi AA. 2006. *Integrated Planning and Control for Convex-bodied Nonholonomic Systems using Local Feedback Control Policies*. In *Robotics: Science and Systems*, vol. 2
24. Reist P, Tedrake R. 2010. *Simulation-based LQR-trees with input and state constraints*. In *2010 IEEE International Conference on Robotics and Automation*, pp. 5504–5510
25. Paxton C, Ratliff N, Eppner C, Fox D. 2019. *Representing Robot Task Plans as Robust Logical-Dynamical Systems*. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5588–5595. ISSN: 2153-0866
26. Ames AD, Coogan S, Egerstedt M, Notomista G, Sreenath K, Tabuada P. 2019. *Control Barrier Functions: Theory and Applications*. In *2019 18th European Control Conference (ECC)*, pp. 3420–3431
27. Ögren P. 2006. *Autonomous UCAV Strike Missions Using Behavior Control Lyapunov Functions*. In *AIAA Guidance, Navigation, and Control*
28. Özkahraman, Ögren P. 2020. *Combining Control Barrier Functions and Behavior Trees for Multi-Agent Underwater Coverage Missions*. In *2020 59th IEEE Conference on Decision and Control (CDC)*, pp. 5275–5282
29. Endsley MR. 2015. Autonomous Horizons: System Autonomy in the Air Force – A Path to the Future. Volume I: Human-AutonomyTeaming. Tech. rep., United States Air Force Office of the Chief Scientist
30. Paxton C, Jonathan F, Hundt A, Mutlu B, Hager GD. 2017. User Experience of the CoSTAR System for Instruction of Collaborative Robots. *arXiv:1703.07890 [cs]*
31. Vinyals O, Babuschkin I, Czarnecki WM, Silver D. 2019. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature* 575(7782):350–354
32. Pereira RdP, Engel PM. 2015. A Framework for Constrained and Adaptive Behavior-Based Agents. *arXiv:1506.02312 [cs]*
33. Kartasev M. 2019. Integrating Reinforcement Learning into Behavior Trees by Hierarchical Composition. Master thesis, KTH Royal Institute of Technology
34. Sprague CI, Ögren P. 2018. Adding Neural Network Controllers to Behavior Trees without Destroying Performance Guarantees. *arXiv:1809.10283 [cs]*
35. Dey R, Child C. 2013. *QL-BT: Enhancing Behaviour Tree Design and Implementation with Q-Learning*. In *2013 IEEE Conference on Computational Inteligence in Games (CIG)*, pp. 1–8. Niagara Falls, ON, Canada: IEEE
36. Fu Y, Qin L, Yin Q. 2016. *A Reinforcement Learning Behavior Tree Framework for Game AI*. In *Proceedings of the 2016 International Conference on Economics, Social Science, Arts, Education and Management Engineering*. Huhhot, China: Atlantis Press
37. Zhang Q, Sun L, Jiao P, Yin Q. 2017. *Combining Behavior Trees with MAXQ Learning to Facilitate CGFs Behavior Modeling*. In *2017 4th International Conference on Systems and Informatics (ICSAI)*, pp. 525–531
38. Zhu X. 2019. Behavior tree design of intelligent behavior of non-player character (NPC) based on Unity3D. *Journal of Intelligent & Fuzzy Systems* 37(5):6071–6079
39. Hannaford B, Hu D, Zhang D, Li Y. 2016. Simulation Results on Selector Adaptation in Behavior Trees. *arXiv:1606.09219 [cs]*

40. Whitley D. 1994. A genetic algorithm tutorial. *Statistics and Computing* 4(2):65–85
41. Rothlauf F, Oetzel M. 2006. *On the locality of grammatical evolution*. In *European Conference on Genetic Programming*, pp. 320–330. Springer
42. Luke S, Panait L. 2006. A Comparison of Bloat Control Methods for Genetic Programming. *Evolutionary Computation* 14(3):309–344
43. Colledanchise M, Parasuraman R, Ögren P. 2019. Learning of Behavior Trees for Autonomous Agents. *IEEE Transactions on Games* 11(2):183–189
44. Hallawa A, Schug S, Iacca G, Ascheid G. 2020. *Evolving Instinctive Behaviour in Resource-Constrained Autonomous Agents Using Grammatical Evolution*. In *Applications of Evolutionary Computation*, ed. PA Castillo, JL Jiménez Laredo, F Fernández de Vega, pp. 369–383, Lecture Notes in Computer Science, pp. 369–383. Cham: Springer International Publishing
45. Lim CU, Baumgarten R, Colton S. 2010. Evolving Behaviour Trees for the Commercial Game DEFCON. In *Applications of Evolutionary Computation*, ed. D Hutchison, GN Yannakakis, pp. 100–110, vol. 6024. Springer Berlin Heidelberg
46. Nicolau M, Perez-Liebana D, O'Neill M, Brabazon A. 2017. Evolutionary Behavior Tree Approaches for Navigating Platform Games. *IEEE Transactions on Computational Intelligence and AI in Games* 9(3):227–238
47. Iovino M, Styrud J, Falco P, Smith C. 2020. Learning Behavior Trees with Genetic Programming in Unpredictable Environments. *arXiv:2011.03252 [cs]* ArXiv: 2011.03252
48. Paduraru C, Paduraru M. 2019. *Automatic Difficulty Management and Testing in Games Using a Framework Based on Behavior Trees and Genetic Algorithms*. In *arXiv:1909.04368 [Cs]*
49. Styrud J, Iovino M, Norrlöf M, Björkman M, Smith C. 2021. Combining Planning and Learning of Behavior Trees for Robotic Assembly. *arXiv:2103.09036 [cs]* ArXiv: 2103.09036
50. Neupane A, Goodrich M. 2019. *Learning Swarm Behaviors Using Grammatical Evolution and Behavior Trees*. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, pp. 513–520. Macao, China: International Joint Conferences on Artificial Intelligence Organization
51. Jones S, Studley M, Hauert S, Winfield A. 2018. Evolving Behaviour Trees for Swarm Robotics. In *Distributed Autonomous Robotic Systems: The 13th International Symposium*, ed. R Groß, A Kolling, S Berman, E Frazzoli, A Martinoli, F Matsuno, M Gauci, pp. 487–501, Springer Proceedings in Advanced Robotics. Cham: Springer International Publishing
52. Martín F, Morelli M, Espinoza H, Lera FJR, Matellán V. 2021. Optimized Execution of PDDL Plans using Behavior Trees. *arXiv:2101.01964 [cs]*
53. Colledanchise M, Almeida D, Ögren P. 2019. *Towards Blended Reactive Planning and Acting using Behavior Trees*. In *IEEE Int. Conference on Robotics and Automation*. Montreal, Canada: IEEE
54. Tadewos TG, Shamgah L, Karimoddini A. 2019. *Automatic Safe Behaviour Tree Synthesis for Autonomous Agents*. In *2019 IEEE 58th Conference on Decision and Control (CDC)*, pp. 2776–2781
55. Tadewos TG, Shamgah L, Karimoddini A. 2019. *On-the-Fly Decentralized Tasking of Autonomous Vehicles*. In *2019 IEEE 58th Conference on Decision and Control (CDC)*, pp. 2770–2775
56. Zhou H, Min H, Lin Y. 2019. *An Autonomous Task Algorithm Based on Behavior Trees for Robot*. In *2019 2nd*

China Symposium on Cognitive Computing and Hybrid Intelligence (CCHI), pp. 64–70

57. Paxton C, Ratliff N, Eppner C, Fox D. 2019. *Representing Robot Task Plans as Robust Logical-Dynamical Systems*. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5588–5595
58. Schwab P, Hlavacs H. 2015. *Capturing the Essence: Towards the Automated Generation of Transparent Behavior Models*. In *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*
59. Kuckling J, Ligot A, Bozhinoski D, Birattari M. 2018. Behavior Trees as a Control Architecture in the Automatic Modular Design of Robot Swarms. In *Swarm Intelligence*, ed. M Dorigo, pp. 30–43, Lecture Notes in Computer Science. Springer International Publishing
60. Neufeld X, Mostaghim S, Brand S. 2018. *A Hybrid Approach to Planning and Execution in Dynamic Environments Through Hierarchical Task Networks and Behavior Trees*. In *Fourteenth Artificial Intelligence and Interactive Digital Entertainment Conference*
61. Rovida F, Grossmann B, Krüger V. 2017. *Extended Behavior Trees for Quick Definition of Flexible Robotic Tasks*. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6793–6800
62. Segura-Muros JÁ, Fernández-Olivares J. 2017. *Integration of an Automated Hierarchical Task Planner in ROS Using Behaviour Trees*. In *2017 6th International Conference on Space Mission Challenges for Information Technology (SMC-IT)*, pp. 20–25
63. Hölzl M, Gabor T. 2015. Reasoning and Learning for Awareness and Adaptation. In *Software Engineering for Collective Autonomic Systems: The ASCENS Approach*, ed. M Wirsing, M Hölzl, N Koch, P Mayer, pp. 249–290, Lecture Notes in Computer Science. Cham: Springer International Publishing
64. Colledanchise M, Murray RM, Ögren P. 2017. *Synthesis of Correct-by-Construction Behavior Trees*. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6039–6046
65. Lan M, Lai S, Lee TH, Chen BM. 2019. *Autonomous Task Planning and Acting for Micro Aerial Vehicles*. In *2019 IEEE 15th International Conference on Control and Automation (ICCA)*, pp. 738–745
66. Biggar O, Zamani M. 2020. A Framework for Formal Verification of Behavior Trees with Linear Temporal Logic. *IEEE Robotics and Automation Letters*