RESEARCH-ARTICLE

BT2Automata: Expressing Behavior Trees as Automata for Formal Control Synthesis

RYAN MATHEU, University of Maryland, College Park, College Park, MD, United States

ANIRUDDH G. PURANIC, University of Maryland, College Park, College Park, MD, United States

JOHN S. BARAS, University of Maryland, College Park, College Park, MD, United States

CALIN BELTA, University of Maryland, College Park, College Park, MD, United States

Published: 21 May 2025

Citation in BibTeX format

HSCC '25: Computation and Control
May 6 - 9, 2025
CA, Irvine, USA

Conference Sponsors:
SIGBED

HSCC '25: Proceedings of the 28th ACM International Conference on Hybrid Systems: Computation and Control (May 2025)
https://doi.org/10.1145/3716863.3718042
ISBN: 9798400715044

BT2Automata: Expressing Behavior Trees as Automata for Formal Control Synthesis

Ryan Matheu
rmatheu@umd.edu
University of Maryland
College Park, Maryland, USA

Aniruddh G. Puranic
puranic@umd.edu
University of Maryland
College Park, Maryland, USA

John S. Baras
baras@umd.edu
University of Maryland
College Park, Maryland, USA

Calin Belta
calin@umd.edu
University of Maryland
College Park, Maryland, USA

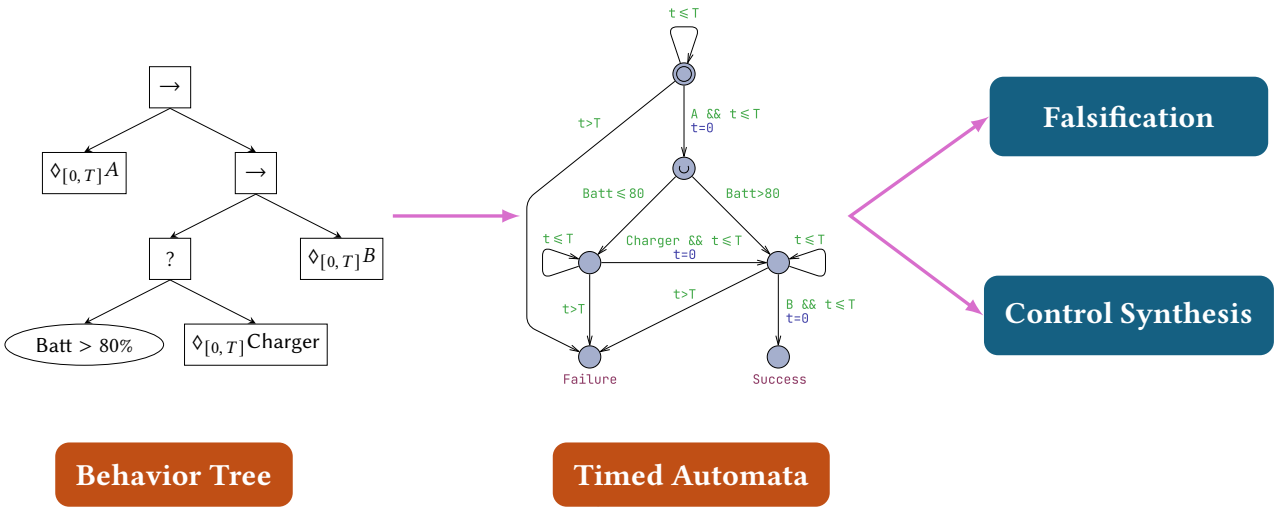


Figure 1: Overview of the BT2Automata Framework.

Abstract

This research presents a novel approach to bridging the gap between the interpretable and flexible nature of Behavior Trees (BTs) and the rigorous formal verification and synthesis capabilities of temporal logics. Temporal logics, such as Linear Temporal Logic (LTL) and Metric Interval Temporal Logic (MITL), are widely used for task specification due to their intuitive syntax for expressing temporally evolving behaviors. However, encoding complex task dependencies and recovery actions in temporal logic can lead to intractability. BTs, known for their modular structure and dynamic adaptability, have gained popularity in robotics for task specification. Despite the advantages of BTs, their flexible structure complicates formal analysis for safety and performance guarantees, limiting their use in control synthesis. This work presents a novel approach by translating BTs into Timed Automata (TA), thus enabling falsification (counterexample generation) with Uppaal to identify inconsistencies and ensure language completeness, especially when defined with timing constraints. This integration allows for the detection of potential inconsistencies, the monitoring of temporal properties, and the synthesis of automaton and sampling based control strategies that guarantee satisfaction of task objectives.

CCS Concepts

• **Theory of computation** → **Timed and hybrid models**; **Formal languages and automata theory**; • **Computing methodologies** → **Planning and scheduling**; **Control methods**.

Keywords

Behavior Trees, Timed Automata, Temporal Logic, Control Synthesis, Falsification

HSCC '25, Irvine, CA, USA

ACM ISBN 979-8-4007-1504-4/2025/05
https://doi.org/10.1145/3716863.3718042

ACM Reference Format:
Ryan Matheu, Aniruddh G. Puranic, John S. Baras, and Calin Belta. 2025. BT2Automata: Expressing Behavior Trees as Automata for Formal Control Synthesis. In *28th ACM International Conference on Hybrid Systems: Computation and Control (HSCC '25), May 6–9, 2025, Irvine, CA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3716863.3718042

1 Introduction

With the increase in the capabilities of robotic and cyber-physical systems (CPS), so has the complexity of specifying task plans, particularly those with explicit timing requirements. Temporal logics, such as Linear Temporal Logic (LTL) and Metric Interval Temporal Logic (MITL), have become a prominent specification formalism for robotic tasks due to their intuitive grammar and ability to reason about temporally-evolving behaviors [18, 21, 32, 33, 38]. However, temporal logic-based motion and task planning can quickly become intractable when attempting to encode recovery behaviors and interdependencies between tasks.

Behavior Trees (BTs) [15], initially employed in computer games to specify behaviors for animated characters, have recently gained prominence in robotics due to their ease of use and graphical nature for designing task plans and control software. BTs are designed to be dynamic, allowing for modular changes to the tree structure without extensive reconfiguration. This adaptability is crucial for enabling designers to respond effectively to changes in their environment or task requirements. BTs are tightly integrated in ROS2 [31] and have open source implementations in both C++ and Python [20, 45]. BTs have become the preferred tool as compared to traditional control structures like Hierarchical Finite State Machines [44] due to their increase in modularity. BTs can be constructed as standalone controllers, task specifications, and more recently temporal logic specifications [42]. The dynamic/modular nature of BTs poses a major challenge to formalizing them in order to reason about safety and performance guarantees, and hence using them for downstream objectives such as control synthesis.

This work extends the notion of using BTs as temporal logic task specifications and presents a novel translation from BTs to Timed Automata (TA) [9]. An equivalent TA representation of a BT allows for the use of model checkers, in particular Uppaal [24], to check for inconsistencies, language emptiness and falsify the models. Furthermore, the TA representation allows for automaton-based control synthesis via planners and sampling-based methods such as reinforcement learning.

Our main contributions in this work are:

(1) We propose mechanisms to express BT nodes as TA and develop a hierarchical logic to monitor the control and execution nodes of a BT respectively.
(2) By defining the qualitative semantics for the hierarchical logic, we show the language equivalence between a BT and its corresponding TA.
(3) We show how our framework can be used for generating runs that satisfy or violate task specifications using Uppaal and also synthesizing control for robotic tasks.

The structure of the paper is as follows: Section 2 describes works related to the use of BTs for planning and control, and the use of BTs as temporal logic task specifications. Section 3 introduces the necessary concepts from formal methods, automata theory, and temporal logic. Section 4 describes the problem definition in detail. Section 5 explains the translation from BTs to TA. Section 6 explains the composition of a BT equivalent TA with leaf nodes derived from MITL specifications, their embedding in Uppaal, and also employs control synthesis for a mobile robot and a robot manipulation task.

2 Related Works

The concept of BTs arises in two places within the engineering and computer science domains. The first concept was as a formal systems engineering tool for capturing functional requirements [19]. The authors in [16, 23] extend this concept to include system requirements with explicit timing constraints. The second concept, recently popularized in [15], defines a BT as a structure for programming controllers with reactive switching for robotic tasks. This work focuses on and builds upon the second concept of BTs.

There have been several attempts to formalize BTs and synthesize controls. The authors of [43] propose BehaVerify, a tool for model checking BTs. This tool converts a BT defined using PyTrees to a nuXmv model that can be used for model-checking the BT with respect to LTL specifications. Another work by the authors of [13] aims to synthesize correct-by-construction BTs that are guaranteed to satisfy the LTL formulas. In both these works, the characteristics of a BT (i.e. structure and composition of nodes) are expressed via a large number of explicitly designed symbolic rules, which are time-consuming and reduce human-interpretability. As BTs are inherently designed to be dynamic/mutable, their equivalent symbolic representations proposed in these works may require the translation process to restart from the BT root. Furthermore, both these methods are limited to specifications expressed in LTL (either the full language or its fragments). Hence they do not account for any timing or state-based guard conditions during BT execution. In addition, the BehaVerify tool is agnostic to control models such as neural networks and abstracts the corresponding execution nodes to intervals of output values, which can create loose approximations of the bounds. In our work, we explicitly represent BTs as TA thus allowing us to reason and monitor both control and execution nodes of the BT accurately.

The authors in [42] introduce the notion of utilizing BTs as task specifications through Temporal BTs (TBTs), which is an offline temporal logic monitor. This extends the widely popular Signal Temporal Logic (STL) to include BT-inspired operators. A TBT formula is a BT whose leaf nodes are STL formulas. TBT formulas encode the desirable structures of BTs, and are equipped with quantitative semantics that indicate the degree of satisfaction (robustness) of a trace with respect to the TBT formula. The robustness measure is determined by: (1) segmenting a trace into subtraces that are mapped to unique paths of the corresponding BT, (2) computing the STL-based robustness of the subtraces, and (3) propagating the subtrace robustness through to the root of the BT. This formalism is well-suited for offline monitoring for open-loop control synthesis. However, feedback/closed-loop control mechanisms require intermediate feedback, thus eliciting the need for automata-based and online variants of TBTs.

Temporal logics have been widely investigated in the context of control synthesis. The literature in this domain utilize the partial-trace quantitative semantics of temporal logics such as STL and the automata induced by LTL or MITL to describe and shape reward functions in motion planning and reinforcement learning (RL) tasks [2, 8, 26, 29, 30]. However, synthesizing control behaviors from automata is highly non-trivial as it can be resource-intensive [12, 32]. One of the ways to improve tractability is by the use of expert

demonstrations of desired system behaviors. Human demonstrations have been used alongside temporal logic task specifications for direct imitation learning or in the context of Inverse RL to learn reward functions and hence the control policy that satisfies the specifications [1, 25, 28, 39].

3 Background

In this section we review the necessary concepts from automata theory and temporal logics to aid the reader in understanding the translation from BTs to TA.

Definition 3.1. Let AP be a finite set of atomic propositions. A propositional variable $p \in AP$ is associated with a set $X_p \subset X \subseteq \mathbb{R}^n$ such that when a state $x \in X_p$, proposition p is taken to be True ($\top$), and False ($\bot$) otherwise.

Let the 2D workspace for a mobile robot be discretized into a grid of uniform rectangular cells. Transitions are allowed between free adjacent cells and are assumed to incur a fixed time penalty faithful to the dynamics of the robot. Each cell has associated with it a set of propositional variables which represent the truth of certain system properties. These properties may include, but are not limited to, goals, objects, and obstacles. Control synthesis using formal methods and environments represented as discrete transition systems is explored in [37, 41, 48]. We represent the interaction of a mobile robot with its environment as a *labeled transition system*.

Definition 3.2 (Labeled Transition System). A Labeled Transition System (LTS) is a tuple $\mathcal{W} = (AP, V, v_0, E, \mathcal{L})$ where AP is a finite set of atomic propositions, (V, E) is an undirected graph, $v_0 \in V$ is an initial state, and $\mathcal{L} : V \to 2^{AP}$ is a labeling function.

The trajectory of a mobile robot in the discretized workspace produces a *timed run* of states that can be mapped to subsets of AP via the labeling function. Such a sequence is referred to as a *timed word*.

Definition 3.3 (Timed Run of an LTS). A timed run of an LTS $\mathcal{W}$ is a pair (v, t) such that $v = v^0 v^1 v^2 \cdots$ is a state sequence with $v^i \in V$, and $t = t^0 t^1 t^2 \cdots$ is a monotonic time sequence with $t^i \in \mathbb{R}^{\geq 0}$ and $t^0 < t^1 < t^2 < \cdots$.

Definition 3.4 (Timed Word). A timed word x over the set AP is a pair (w, t) such that $w = \sigma^0 \sigma^1 \sigma^2 \cdots$ is an untimed word with $\sigma^i \in 2^{AP}$, and $t = t^0 t^1 t^2 \cdots$ is a monotonic time sequence with $t^i \in \mathbb{R}^{\geq 0}$ and $t^0 < t^1 < t^2 < \cdots$. Furthermore, a timed run (v, t) of an LTS can be mapped to a timed word via the labeling function such that $x = (w, t)$ where $w = \mathcal{L}(v^0)\mathcal{L}(v^1)\mathcal{L}(v^2)\cdots$.

Metric Interval Temporal Logic (MITL), first introduced in [5] as a decidable fragment of Metric Temporal Logic (MTL), is a temporal logic interpreted over Boolean signals. MITL constrains its temporal operators to reason over *time intervals*.

Definition 3.5 (Metric Interval Temporal Logic). A Metric Interval Temporal Logic (MITL) formula over a set of atomic propositions AP is defined by the grammar:

$$\varphi := p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \Box_I \varphi \mid \Diamond_I \varphi \mid \varphi_1 \mathcal{U}_I \varphi_2$$

where $p \in AP$, $\wedge$ and $\vee$ are logical conjunction and disjunction, respectively; $\Box$, $\Diamond$, and $\mathcal{U}$ are the always, eventually, and until temporal operators, respectively; and $I \subseteq \mathbb{R}^{\geq 0}$ is an interval of the form $[a, b]$, $b > a \geq 0$.

The satisfiability of MITL formulas is decidable in EXPSPACE. For fragments that use intervals only of the form $[0, T]$, it is decidable in PSPACE [5]. The primary benefit of MITL is that for all MITL formulas, there exists a timed Büchi automaton that accepts timed words that model the formula [33]. A compositional algorithm for translating MITL to TA is demonstrated in [11].

Definition 3.6 (Timed Büchi Automaton). A Timed Büchi Automaton (TBA) [4] is a tuple $\mathcal{A} = (\Sigma, Q, q_0, C, E, F)$ where Σ is the finite input alphabet, Q is a finite set of states, $q_0 \in Q$ is the initial state, C is a finite set of clocks, $E \subseteq Q \times \Sigma \times \mathcal{B}(C) \times 2^C \times Q$ is the set of edges, and $F \subset Q$ is the set of accepting states. The set $\mathcal{B}(C)$ represents the clock constraints on clocks $c \in C$ of the form $c \sim x$, where $\sim \in \{<, \leq, >, \geq, ==, !=\}$ and $x \in \mathbb{N}$. An edge $(q_1, g, b, r, q_2) \in E$ represents a transition from state q_1, q_2 with guard g, clock constraints b, and clock resets r.

3.1 Behavior Trees

BTs [15] are directed acyclic graphs consisting of execution and control nodes. Each node in the tree has exactly one parent node with the exception of the root node having no parent. Leaf nodes are *execution* nodes, responsible for observing or changing the state of the underlying system. Connecting the execution nodes and responsible for the structure of the tree are *control* nodes. The control nodes determine the execution flow of the tree and return conditions based on the status of their children. Each node in a BT has three possible conditions: *Success*, *Failure*, and *Running*. The control node types are:

Sequence Node (→): The *Sequence* node can have $N \geq 1$ children. The Sequence node executes children from left-to-right contingent on each child returning *Success*. The Sequence node returns *Success* if all children succeed, *Failure* if a single child fails, and *Running* otherwise.

Selector Node (?): The *Selector* node can have $N \geq 1$ children. The Selector is the complement of the Sequence node. The Selector node executes children from left-to-right contingent on each child returning *Failure*. The Selector node returns *Failure* if all children fail, *Success* if a single child succeeds, and *Running* otherwise.

Parallel Node (⇉): The *Parallel* node can have $N \geq 1$ children. The Parallel node executes all children in parallel. The Parallel node returns *Success* if M children succeed where $1 \leq M \leq N$ is a user-defined parameter. The Parallel node returns *Failure* if K children fail where $1 \leq K \leq N - M + 1$ is a user-defined parameter. Otherwise the Parallel node returns *Running*.

Execution nodes are further classified into two types: *action* and *condition* nodes. Actions nodes are represented graphically with a rectangle. Action nodes drive the state of the underlying system and return a logical status of *Running* while executing. When an action node terminates, it must return a status of *Success* or *Failure*. Condition nodes are represented graphically with an ellipse. Condition nodes query the state of the underlying system and instantly return a logical status, either *Success* or *Failure*.

Definition 3.7 (TBT Syntax and Semantics [42]). Let φ be a MITL formula over AP. A TBT is defined by the following grammar:

$$\mathcal{T} := \text{Leaf}(\varphi) \mid \text{Seq}(\mathcal{T}_1, \ldots, \mathcal{T}_n) \mid \text{Sel}(\mathcal{T}_1, \ldots, \mathcal{T}_n) \mid \text{Par}_M^K(\mathcal{T}_1, \ldots, \mathcal{T}_n)$$

Given a timed word x over AP, the satisfaction of a TBT specification is defined inductively:

- $x \models \text{Leaf}(\varphi) \Leftrightarrow x \models \varphi$
- $x \models \text{Seq}(\mathcal{T}_1, \ldots, \mathcal{T}_n) \Leftrightarrow \exists\, i_1 < i_2 < \cdots < i_{n-1}$ s.t. $x^0 \cdots x^{i_1} \models \mathcal{T}_1$, $x^{i_1+1} \cdots x^{i_2} \models \mathcal{T}_2, \ldots, x^{i_{n-1}+1} \cdots \models \mathcal{T}_n$
- $x \models \text{Sel}(\mathcal{T}_1, \ldots, \mathcal{T}_n) \Leftrightarrow \exists\, i \geq 0,\ j \in \{1, \ldots, n\}$ s.t. $x^i \cdots \models \mathcal{T}_j$
- $x \models \text{Par}_M^K(\mathcal{T}_1, \ldots, \mathcal{T}_n) \Leftrightarrow \exists\, \{i_1, \ldots, i_M\} \subset \{1, \ldots, n\}$ s.t. $x \models \mathcal{T}_{i_1}, \ldots, x \models \mathcal{T}_{i_M}$

3.2 UPPAAL and Networks of TA

Uppaal [9] is a toolbox for the design and verification of real time systems modeled as Networks of TA (NTA). Uppaal extends TA with several features. The features of importance for this work are Boolean and integer variables, synchronization channels, and urgent locations.

Boolean and integer variables can be in the global scope and be referenced by any TA in the network, or be internal and accessible by the owner only. Guard conditions on transitions are extended to include binary evaluations on integer variables of the form $x_1 \sim x_2$ where $\sim \in \{<, \leq, >, \geq, ==, !=\}$. Synchronizations allow simultaneous transitions between two or more automata. An automaton transitioning along an edge with the label `e!` will synchronize and force a transition of another automata in the network with the label `e?` so long as the guard conditions are satisfied. Finally, urgent locations are states that do not allow for the passage of time. A system in an urgent location will freeze time until an outgoing transition occurs. Urgent locations are syntactically convenient for modeling intermediate states in non-blocking processes.

TA in Uppaal are constructed using *templates*. Templates are reusable graphical definitions of individual TA. A system declaration is created by instantiating one or more templates with a unique name and (possibly) parameters. An NTA in Uppaal is a system of one or more TA operating concurrently and communicating over shared variables and synchronizations.

Definition 3.8 (Timed Automata Extended With Integer Variables). A TA extended with integer variables is a tuple $(\Sigma, \Gamma, Q, q_0, C, N, E, F)$ where Σ is the finite input alphabet, Γ is the finite output alphabet, Q is a finite set of states, $q_0 \in Q$ is an initial state, C is a finite set of clocks, N is a finite set of integer variables, $E \subseteq Q \times \Sigma \times \mathcal{B}(C) \times \mathcal{H}(N) \times 2^N \times Q \times \Gamma$ is the set of edges, and $F \subset Q$ is the set of accepting states. $\mathcal{B}(C)$ is the set of clock constraints over the set C. $\mathcal{H}(N)$ is the set of Boolean constraints involving integers from N of the form $n_1 \sim n_2$ or $n_1 \sim z$ where $\sim \in \{<, \leq, >, \geq, ==, !=\}$ and $n_1, n_2 \in N$ and $z \in \mathbb{Z}$. These along with valuations of the input determine the guard conditions on transitions. Furthermore, edges are extended to include deterministic assignments on integer variables of the form $n \star z$ where $\star \in \{+=, -=, =\}$, $n \in N$ and $z \in \mathbb{Z}$, representing the increment, decrement, and assignment operators over integers respectively.

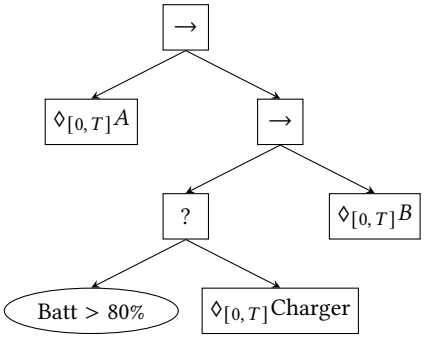


Figure 2: An example TBT for a mobile robot.

4 Problem Definition and Approach

Consider a robot-environment interaction represented as an LTS $\mathcal{W}$ over a finite set of atomic propositions AP. Let $\mathcal{T}$ represent a task specification encoded as a TBT formula whose leaf nodes are bounded MITL formulas over AP.

Problem 1. *Synthesize a timed run over $\mathcal{W}$ such that the timed word $x = (w, t)$ associated with the timed run satisfies the TBT specification, $x \models \mathcal{T}$.*

Problem 1 requires the development of a formal model of TBTs with MITL leaf nodes. The existence of TA that model MITL formula and the similarities between BTs and finite state machines [15] indicates that a suitable formal model will be a type of automaton. Furthermore, the embedding of BTs as a type of automaton enables the use of numerous model-checking tools for finding temporal logic-satisfying runs (or counterexamples) and formal control synthesis.

5 Encoding BTs as TA

In this section we introduce a mapping between BTs and TA. We start by thoroughly describing how a BT can formally model a task specification. We then show that a BT can be represented as a system consisting of a hierarchical logic where:

(1) the *inner logic*, specified via MITL, monitors the execution of leaf nodes, and
(2) the *outer logic*, dependent on the structure of the BT, monitors the traversal of the tree.

5.1 BTs as Specifications

TBTs allow for rich task specification using the intuitive graphical structure of BTs and provide several advantages over traditional temporal logics. Sequential tasks and recovery behaviors are easier to specify compared to a single temporal logic formula which may require deep nesting to get the desired behavior. Consider the following mobile robot task specification: *Reach goal A within T time units; then if the battery is less than or equal to* 80%*, return to the charging station within T time units; then proceed to goal B within T time units.* The *if–else* condition can be challenging to specify using MITL directly. Instead, the BT Selector node allows one to specify conditional expressions clearly and concisely. The equivalent TBT is shown in Figure 2.

In this paper, we consider TBTs whose leaf nodes consist of MITL formulas over a finite set of atomic propositions. Furthermore, we restrict leaf nodes to bounded formulas whose satisfaction can be determined in finite time. This follows from the reactivity of BTs in that leaf nodes need to be capable of reaching a success or failure state in finite time.

Example 5.1. Let AP be a finite set of atomic propositions. If $A \in AP$ is a propositional variable, then the formula $\Diamond_{[0,T]}A$ is decidable in at most T time units. If $x = (w, t)$ is a timed word over AP, then the formula is satisfied at the first instance $i \in \mathbb{N}^{\geq 0}$ such that $A \in w^i$ and $t^i \leq T$. If $A \notin w^i$ for all $i \in \mathbb{N}^{\geq 0}$, $t^i \leq T$, then the formula is violated.

We define condition nodes of a TBT as formulas over the following grammar:

$$\varphi := p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2$$

We restrict condition nodes to untimed logical connectives such that the satisfaction of a condition node is decidable in zero time.

Example 5.2. If $A, B \in AP$, then $A \wedge B$ and $\neg A \wedge B$ are valid condition node formulas.

Due to the three-state nature of the return status of BT nodes, we adopt a balanced ternary (three-valued) logic with truth values *Success* (True, $\top \mapsto +1$), *Running* (Unknown, $\varnothing \mapsto 0$), and *Failure* (False, $\bot \mapsto -1$). Let $X \in \{Failure, Running, Success\}$ (abbr. $\{F, R, S\}$), the qualitative semantics of the three-valued logic are defined in Table 1. It should be noted that the inner logic is case specific. The qualitative semantics of the inner logic is determined by the qualitative semantics of the temporal logic representing the leaf nodes of the BT.

Table 1: Ternary logic qualitative semantics.

Operation	Qualitative	
Negation	$\neg S = F$	$\neg 1 = -1$
	$\neg F = S$	$\neg -1 = 1$
	$\neg R = R$	$\neg 0 = 0$
Conjunction	$F \wedge X = F$	$\min(X_1, X_2)$
	$S \wedge X = X$	
	$R \wedge \{S, R\} = R$	
	$R \wedge F = F$	
Disjunction	$S \vee X = S$	$\max(X_1, X_2)$
	$R \vee \{F, R\} = R$	
	$R \vee S = S$	
	$F \vee F = F$	
	$F \vee R = R$	

Definition 5.3 (BT Inner Logic Over MITL). Let φ be an MITL leaf node formula. The ternary valuation v_φ of the formula φ given a timed word x is mapping $v_\varphi : (2^{AP} \times \mathbb{R}^{\geq 0})^* \to \{F, R, S\}$ defined as:

$$v_\varphi(x) = \begin{cases} S, & x \models \varphi, \\ F, & x \not\models \varphi, \\ R, & \text{otherwise.} \end{cases} \tag{1}$$

This ensures that the inner logic is consistent with the return status of BTs. When the satisfaction of a formula is undecided, the leaf node will return a status of *Running* indicating that the node has not finished executing. The outer logic, derived from the rules of BT traversal, is invariant to choice of inner logic.

5.2 BT Outer Logic Qualitative Semantics

The outer logic for each BT node can be thought of as an input-output mapping with memory. Each control node, Sequence, Selector, and Parallel, takes as input a word over the alphabet $\{F, R, S\}$ determined by the status of its children and produces an output over the same alphabet.

Definition 5.4 (BT Outer Logic). The BT outer logic is a mapping $\chi : \{F, R, S\}^* \to \{F, R, S\}$.

For the Sequence node, the output of the outer logic is: S if all children have succeeded, F if a single child has failed, and R otherwise. Given a word $w \in \{F, R, S\}^*$, the input-output mapping for a Sequence node with N children is defined as:

$$\chi_{\text{seq}}(w) = \begin{cases} S, & S \in w \ N \text{ times}, \\ F, & F \in w, \\ R, & \text{otherwise.} \end{cases} \tag{2}$$

The condition that S must appear in the word N times requires the input-output mapping have memory to retain the number of *Success* inputs. A natural structure for realizing the input-output mapping is an automaton, leading us to the following lemma:

Lemma 5.5. *The outer logic mapping for the Sequence node is realizable by an automaton.*

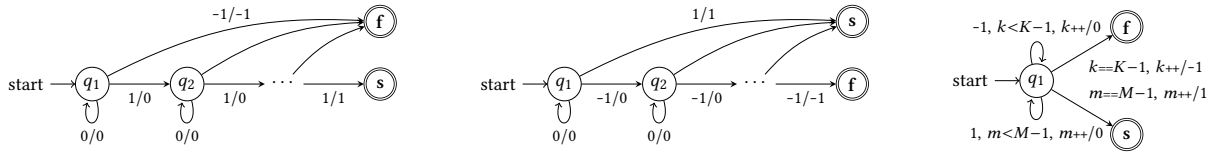
Proof. The Nerode equivalent classes are: $R_1 = \{R^*S\}$, $R_2 = \{w \cdot R^*S \mid w \in R_1\}, \ldots, R_N = \{w \cdot R^*S, \mid w \in R_{N-1}\}$, one class $R_F = \{w \in \{F, R, S\}^* \mid F \in w\}$, and the trivial class $\{R^*\}$. Here "$\cdot$" represents the concatenation operator. For all words w_1, w_2 in an equivalence class and $w \in \{F, R, S\}^*$, we have $\chi_{\text{seq}}(w_1 \cdot w) = \chi_{\text{seq}}(w_2 \cdot w)$. The realization of a Sequence node with N children is therefore an automaton with $N + 2$ states. □

Table 2: Transition and output table for the Sequence node.

δ_{seq}	F	R	S
q_1	**f**	q_1	q_2
q_2	**f**	q_2	q_3
$\vdots$	$\vdots$	$\vdots$	$\vdots$
q_N	**f**	q_N	**s**

y_{seq}	F	R	S
q_1	F	R	R
q_2	F	R	R
$\vdots$	$\vdots$	$\vdots$	$\vdots$
q_N	F	R	S

The equivalent automaton for the Sequence node consists of N intermediate states representing the currently executing child. There are two accepting states, a success state which is reachable when all children return *Success*, and a failure state which is reachable from any intermediate state whenever a *Failure* input is received. See Figure 3(a) for the equivalent outer logic automaton for a Sequence node. We define the transition function $\delta_{\text{seq}} : Q \times \Sigma \to Q$ and the output function $y_{\text{seq}} : Q \times \Sigma \to \Sigma$ for the Sequence node according to Table 2. The outer logic for the Selector node is: F if all children



(a) Sequence node automaton. (b) Selector node automaton. (c) Parallel node automaton.

Figure 3: Equivalent automata for the three BT control nodes.

have failed, S if a single child has succeeded, and R otherwise. Given a word $w \in \{F, R, S\}^*$, the input-output mapping for a selector node with N children is defined as:

$$\chi_{\text{sel}}(w) = \begin{cases} F, & F \in w \ N \text{ times}, \\ S, & S \in w, \\ R, & \text{otherwise}. \end{cases} \tag{3}$$

Lemma 5.6. *The outer logic mapping for the selector node is realizable by an automaton.*

Proof. The Selector node is the complement of the Sequence node. The proof follows exactly from the Sequence node with the only difference being exchanging *Success* and *Failure*. □

We see then that an equivalent representation of a Selector node is an automaton with N intermediate states and two accepting states. The success accepting state is reachable by any intermediate state upon receiving a *Success* input. The failure accepting state is reachable only if all children return *Failure*. See Figure 3(b) for the equivalent automaton representing the outer logic of a Selector node. The Parallel node has success and failure conditions that both depend on choice of M and K. Given a Parallel node with N children, success condition $1 \leq M \leq N$, and failure condition $1 \leq K \leq N - M + 1$, the input-output mapping is defined as:

$$\chi_{\text{par}}(w) = \begin{cases} S, & S \in w \ M \text{ times}, \\ F, & F \in w \ K \text{ times}, \\ R, & \text{otherwise}. \end{cases} \tag{4}$$

For a Parallel node with N children, there are $N \times (N + 1)$ possible valid configurations of M and K, and a traditional automaton encoding of the Parallel node would require $M \times K + 2$ states. To increase readability, we opt for an encoding using automata extended with integer variables. Doing so allows us to represent the superfluous states instead as counters and overall the realization is easier to construct.

Lemma 5.7. *The outer logic mapping of the Parallel node is realizable by an automaton extended with integer variables.*

Proof. By construction. The resulting automaton has three states, one initial state and two accepting states for success and failure. The automaton has two integer variables m and k, both initialized to zero. Self-transitions are enabled while $m < M - 1$ and $k < K - 1$. The self-transition corresponding to a *Success* input increments the variable m. The self-transition corresponding to a *Failure* input increments the variable k. When $m = M - 1$ (respectively $k = K - 1$) a transition to the success state (resp. failure state) is enabled and upon the next *Success* input (resp. *Failure* input), a transition to the accepting state occurs. The automaton accepts words of the form $\{w \in \{F, R, S\}^* \mid S \in w \ M \text{ times}, F \in w \text{ less than } K \text{ times}\}$ and $\{w \in \{F, R, S\}^* \mid F \in w \ K \text{ times}, S \in w \text{ less than } M \text{ times}\}$. See Figure 3(c) for the automaton that realizes the Parallel node. □

The automaton realization can be applied to arbitrary BTs consisting of many layered control nodes. The encoding results in a composite automaton where states representing child nodes are themselves automata.

Theorem 5.8 (BT Recursive Composition). *BT equivalent automata for the Sequence and Selector nodes can be recursively composed creating an equivalent hierarchal automaton for any BT.*

Proof. States (with the exception of success and failure accepting states) in a BT equivalent automaton for the Sequence and Selector node have a unique ingoing transition and two outgoing transitions. Similarly, the equivalent automaton has a unique initial state and exactly two accepting states. Composing nodes together amounts to rewiring the ingoing transition to a state instead to the initial state of the child automaton. The outgoing success transition is rewired to: the next state if the parent is a Sequence node, or the parent's success state if the parent is a Selector node. Similarly the outgoing failure transition is rewired to: the parent's failure state if the parent is a Sequence node, or the next state if the parent is a Selector node. This procedure can be repeated recursively for trees of arbitrary depth. □

We aim to show bisimulation [35] between a BT and its corresponding automata translation. In short, a bisimulation between two LTSs is a binary relation between the states of the transition systems such that for every pair in the relation, there exists another pair that is induced by the transitions of each constituent system for all labels.

Theorem 5.9 (Deriving bisimulation characteristics). *A BT is a bisimulation of its translated automaton.*

Proof. To describe the steps in this proof, we refer to the Figures 3(a), 3(b), 3(c). We first show the bisimilarity between each of the atomic components of a BT and then reason about the composition of these elements into a full BT that describes a task. Recall that the input alphabets are defined over the status of a BT: $\{Running(R), Success(S), Failure(F)\}$. We define $Q_{\mathcal{T}}$ and $Q_{\mathcal{M}}$ to define the states of a BT $\mathcal{T}$ and its corresponding automaton $\mathcal{M}$, respectively. Thus, we define a binary relation $R \subseteq Q_{\mathcal{T}} \times Q_{\mathcal{M}}$, which is the Cartesian product of the two sets of states. Each subtree in a BT is represented by $\mathcal{T}_i$, and its corresponding state in the

automaton by q_i. As shown in Figure 3, each component of a BT produces nodes success **s** and failure **f** in the translated automaton. We define the relations for each type of BT node as follows:

(1) Case 1 - Leaf node: We consider a BT with a single leaf node/subtree $\mathcal{T}$, and define the relation $\mathcal{B} = \{(\mathcal{T}, q), (\mathcal{T}, \mathbf{s}), (\mathcal{T}, \mathbf{f})\}$.
(2) Case 2 - Sequence Node (Fig. 3(a)): Consider a Sequence BT subtree, rooted at *Seq*, with n children subtrees $[\mathcal{T}_1, \cdots, \mathcal{T}_n]$. Define the relation $\mathcal{B} = \{(Seq, q_1), (\mathcal{T}_i, q_i), (\mathcal{T}_i, \mathbf{f}), (Seq, \mathbf{s}), (Seq, \mathbf{f})\}, \forall i = [1, \cdots, n]$.
(3) Case 3 - Selector/Fallback Node (Fig. 3(b)): Consider a Selector BT subtree, rooted at *Fb*, with n children subtrees $[\mathcal{T}_1, \cdots, \mathcal{T}_n]$. Define the relation $\mathcal{B} = \{(Fb, q_1), (\mathcal{T}_i, q_i), (\mathcal{T}_i, \mathbf{s})(Fb, \mathbf{s}), (Fb, \mathbf{f})\}, \forall i = [1, \cdots, n]$.
(4) Case 4 - Parallel Node (Fig. 3(c)): For this node type, note that in a BT, there are two additional parameters M and K, where the node returns *Success* if at least M nodes succeed and *Failure* if at least K nodes fail. In all other cases, it returns *Running*. Since all nodes are running concurrently, they are abstracted into a "meta" node that behaves as a counter for the number of successes and failures as shown in Fig. 3(c). Thus, the Parallel node is reduced to the Case 1 (Leaf node) and its relation is defined similarly, with the Parallel node semantics for the labels.

In all these cases, the initial and final states coincide, i.e., $\mathcal{T}$. We can see that both the initial and final state-pairs are in the relation $\mathcal{B}$. We can enumerate and check that for every label, there exist corresponding transitions in $\mathcal{T}$ and $\mathcal{M}$, the pair of which is contained in $\mathcal{B}$. Thus, $\mathcal{T}$ is a bisimulation of $\mathcal{M}$.

Since we have shown that the automata consisting of BT subtrees as its nodes is a bisimulation of its BT, this implies that the composition of these elementary nodes is also a bisimulation, as each of the elementary nodes can be abstracted like any node in the automaton, and using the relations described above, the bisimulation is deduced. □

5.3 Complexity

The translation for the Sequence/Selector node with N children results in an automaton with $N+2$ nodes. Furthermore, nesting of the Sequence/Selector nodes removes the intermediate success/failure states and rewires the transitions to the global success/failure states. Therefore the equivalent automaton for a BT consisting only of Sequence/Selector nodes will have $N+2$ states where N is the total number of leaf nodes in the BT. The translation for the Parallel node results in an automaton with $M \times K + 2$ states where $1 \leq M \leq N$ and $1 \leq K \leq N - M + 1$. The worst-case number of states is therefore $\max_{M \in \mathbb{N}} -M^2 + (N+1) \times M + 2$ which is second-order in N. For example, a Parallel node with $N = 10$ children encoded as an automaton can have in the worst-case 32 total states amounting to 30 intermediate states and two accepting states. The inner logic, implemented as MITL, suffers from the complexity of the subsequent MITL to TA translation. Typically this is dominated by the until operator. The states required for the translation of a formula $\varphi = p\mathcal{U}_{[a,b]}q$ is exponential in k where $k = \left\lceil \frac{a}{b-a} \right\rceil$. Though recent work demonstrates a translation where the number of states is linear in k [3, 11].

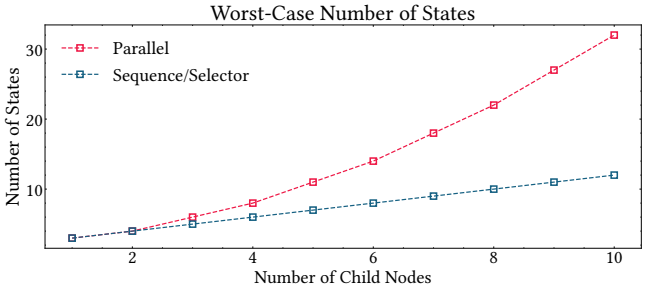


Figure 4: Worst-case number of automaton states for the Parallel and Sequence/Selector nodes with N children.

6 Applications

6.1 Translation to UPPAAL NTA

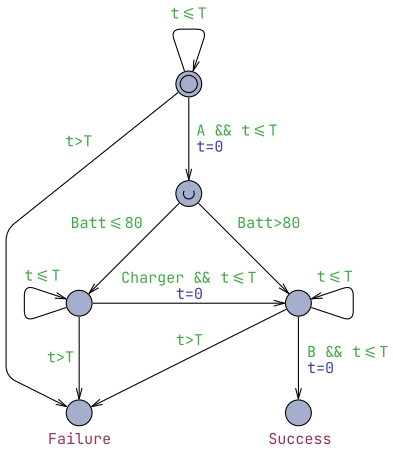


Figure 5: The equivalent UPPAAL TA representation of the TBT in Figure 2.

The translation from a TBT to a network of UPPAAL TA consists of two steps:

(1) the conversion of leaf nodes to timed Büchi automata, and
(2) the composition of leaf automata following the structure of the BT.

Following the convention of BT nodes, each equivalent leaf node automaton must have a single initial state, a single success state, and a single failure state. Similarly, all leaf nodes consisting of bounded temporal logic formulas will have a single, distinct initial, success, and failure state. Composing subtrees via the Sequence, Selector, and Parallel nodes is an exercise in rewiring the subtree automata such that the transitions between terminal states and initial states result in the appropriate transition behavior of the underlying BT as explored in Section 5.

Algorithm 1 describes the process of composing a sequence of one or more subtrees represented as TA. To reduce the number of redundant states, the outgoing success transition of subtree $\mathcal{T}_i$, $i < n$ is connected directly to the initial state of subtree $\mathcal{T}_{i+1}$. All failure transitions connect directly to the single, unique failure

Algorithm 1: `ComposeSequence`$(\mathcal{T}_1, \ldots, \mathcal{T}_n)$

Input: $\mathcal{T}_1, \ldots, \mathcal{T}_n$ – Subtree automata over AP
if $n == 1$ **then**
  return $\mathcal{T}_1$
$Q \leftarrow [], E \leftarrow [], C \leftarrow [], N \leftarrow []$
$q_0 \leftarrow \mathcal{T}_1.init$
$F \leftarrow [\mathcal{T}_n.success, \mathcal{T}_n.failure]$
for $i = 1, \ldots, n$ **do**
  $Q \leftarrow Q + \mathcal{T}_i.states$
  $E \leftarrow E + \mathcal{T}_i.edges$
  $C \leftarrow C + \mathcal{T}_i.clocks$
  $N \leftarrow N + \mathcal{T}_i.vars$
for $i = 1, \ldots, n-1$ **do**
  for $e \in E$ *such that* $e.to = \mathcal{T}_i.success$ **do**
    $e.to \leftarrow \mathcal{T}_{i+1}.init$
  $Q.pop(\mathcal{T}_i.success)$
  for $e \in E$ *such that* $e.to = \mathcal{T}_i.failure$ **do**
    $e.to \leftarrow \mathcal{T}_n.failure$
  $Q.pop(\mathcal{T}_i.failure)$
return (AP, Q, q_0, C, N, E, F)

state of the composed system. Composing the selector node is largely the same process as the Sequence node. The Selector node is the complement of the Sequence node. Children subtrees of the Selector node run in series until the first success. If the execution of a child subtree $\mathcal{T}_i$ results in a failure, the child subtree $\mathcal{T}_{i+1}$ begins executing. Therefore the Selector composition is identical to the Sequence composition with the success and failure states reversed.

$$\texttt{ComposeSelector}(\mathcal{T}_1, \ldots, \mathcal{T}_n) = \texttt{ComposeSequence}(\mathcal{T}_1^c, \ldots, \mathcal{T}_n^c),$$

where $\mathcal{T}_i^c$ is the subtree automaton $\mathcal{T}_i$ with its success and failure states swapped. The challenges of implementing the Parallel node stem from the asynchronous execution of its children subtrees. This challenge is largely subsided when using an NTA communicating over synchronization channels. The automaton representing the Parallel node is constructed in the same manner as Figure 3(c). An outgoing broadcast channel `run!` is placed on the ingoing transition to the initial state of the Parallel node that synchronizes with all child subtrees and initializes their execution. Two synchronization channels `success?` and `failure?` are place on the self transitions to enable the increment of the variables m and k respectively. All child subtrees of the Parallel node are appended with an initial state that can only be left when the synchronization channel `run?` is enabled. Furthermore, each outgoing success and failure transition of each subtree is appended with synchronization channels `success!` and `failure!` respectively to communicate to the Parallel node they have finished executing. The procedure for composing the Parallel node is demonstrated in Algorithm 2.

Example 6.1 (Mobile Robot Navigation). Consider a mobile robot in the grid world show in Figure 6 tasked with a specification given as a TBT as in Figure 2. The mobile robot can traverse locations in the cardinal directions at the expense of 1% of its total battery. The task specification is translated into a Uppaal timed automaton

Algorithm 2: `ComposeParallel`$(\mathcal{T}_1, \ldots, \mathcal{T}_n)$

Input: $\mathcal{T}_1, \ldots, \mathcal{T}_n$ – Subtree automata over AP
Input: M, K – Success/Failure conditions
if $n == 1$ **then**
  return $\mathcal{T}_1$
for $i = 1, \ldots, n$ **do**
  $q_{\text{init}} = \texttt{Location}(init = \texttt{True})$
  $e_{\text{init}} = \texttt{Edge}(q_{\text{init}}, \mathcal{T}_i.locations[\texttt{init}], sync = \texttt{start?})$
  $\mathcal{T}_i.locations \leftarrow \mathcal{T}_i.locations + q_{\text{init}}$
  $\mathcal{T}_i.edges \leftarrow \mathcal{T}_i.edges + e_{\text{init}}$
  $\mathcal{T}_i.edges[\texttt{success}].sync = \texttt{success!}$
  $\mathcal{T}_i.edges[\texttt{failure}].sync = \texttt{failure!}$
$(AP, Q, q_0, C, N, E, F) \leftarrow \texttt{Parallel}(M, K)$ `// Fig. 3(c)`
return $(AP, Q, q_0, C, N, E, F), \mathcal{T}_1, \ldots, \mathcal{T}_n$

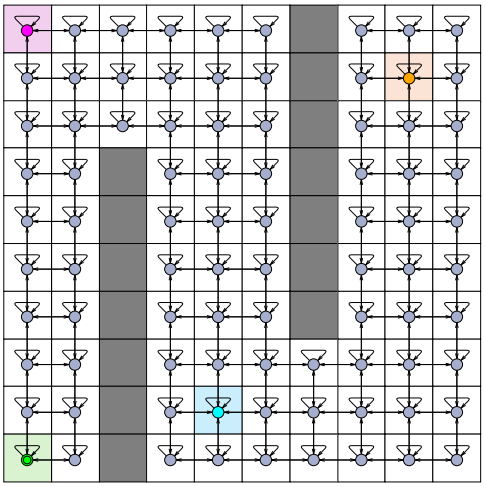
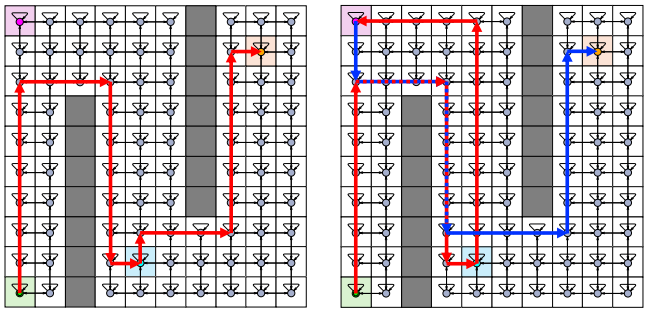


Figure 6: The transition system for the mobile robot example. Here the green location represents the starting position of the robot, the magenta location represents the charging station, the blue location represents the goal A, and the orange location represents the goal B. The two open columns represent walls that the mobile robot is unable to traverse.

(Fig. 5) and a satisfying run is one that results in a transition to the Success state. The synthesized trajectories for 1) a mobile robot with initially 100% battery and 2) a mobile robot with initially 75% batter are show in Figure 7.

6.2 Control Synthesis Case Study: Pick-and-Place Robot Manipulation

We consider the task of a Franka Panda robot that is required to pick up a cube on a table (Fig. 9) and place it at the desired location [22]. The atomic motions needed to perform this task are: (i) move robot end-effector near the cube, (ii) grasp the cube with the end-effector, and (iii) while grasped, move the cube to the target location. We can express the sequence of these atomic motions in a BT (Fig. 8) with the action nodes specifications given by:



(a) Trajectory for a mobile robot with 100% starting battery (able to reach both goals in one go).

(b) Trajectory for a mobile robot with 75% starting battery (needs to recharge before navigating to B).

Figure 7: Synthesized trajectories for mobile robot navigation with BT task specification.

(1) Reaching the cube within a certain tolerance:
$\varphi_{\text{reach\_object}} := \mathbf{FG}(\left\|ee_{pose} - cube_{pose}\right\| \leq 0.05)$.
(2) Grasping the cube (i.e., closing the gripper):
$\varphi_{\text{grasp\_object}} := \mathbf{F}(\theta_{\text{gripper}} \leq 0.1)$.
(3) Placing the cube at the target pose within a certain tolerance:
$\varphi_{\text{move\_obj2target}} := \mathbf{F}(\left\|cube_{pose} - target_{pose}\right\| \leq 0.05)$.

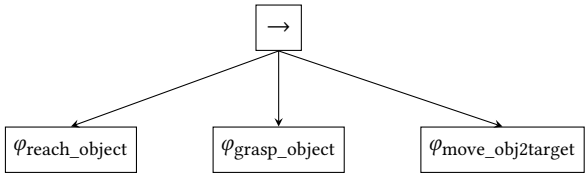


Figure 8: TBT specifying a sequence of subtasks (reach object, grasp it, move it) at a target location for a pick-and-place robot manipulator task. The root of the BT is a Sequence execution node and the 3 subtasks are action nodes that.

The BT composed from the sequence of these actions for this task is given in Fig. 8. We then employ BT-automata to guide the synthesis of a control policy via reinforcement learning (RL). The `rtamt` library [36] is used to define and evaluate the inner logic in STL.

Episode execution. During each episode, the robot starts from the root (Sequence →) node, and is tasked with learning the action nodes from left to right. Each action node is an STL formula that generates the robustness of the partial episode or trace (i.e., from timestep 0 till current timestep t) based on how well the trace satisfies the formula corresponding to that action node. This scalar robustness is used as the reward signal for RL. Once the action nodes are completed, the robustness and hence rewards get accumulated that the RL maximizes to satisfy the BT. Due to the nature of the BT-automata of a Sequence → node, at any step during an episode, failure of any action node in this sequence automatically incurs a negative robustness and hence negative rewards. Thus, this ensures that the entire task succeeds if all the action nodes in the BT are completed successfully in sequence.

This experiment is performed on an Ubuntu 20.04 desktop computer with AMD Ryzen 7 3700X 8-Core CPU and NVIDIA GeForce RTX 2070 Super GPU. The training of the RL agent was completed within 3 hours, and the training curves are shown in Fig. 10. The agent was trained using TQC [27] for $3 \cdot 10^6$ timesteps, which resulted in a success rate of 99%. The learned model was evaluated over 5 random seeds with 20 test trials for each seed, thus totaling 100 test trials. The model achieved an average 100% success rate on the test trials. A sample trajectory of the automata-guided learned model is shown in Fig. 11.

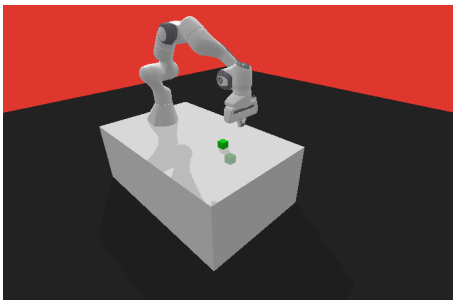


Figure 9: Panda Pick-and-Place Environment.

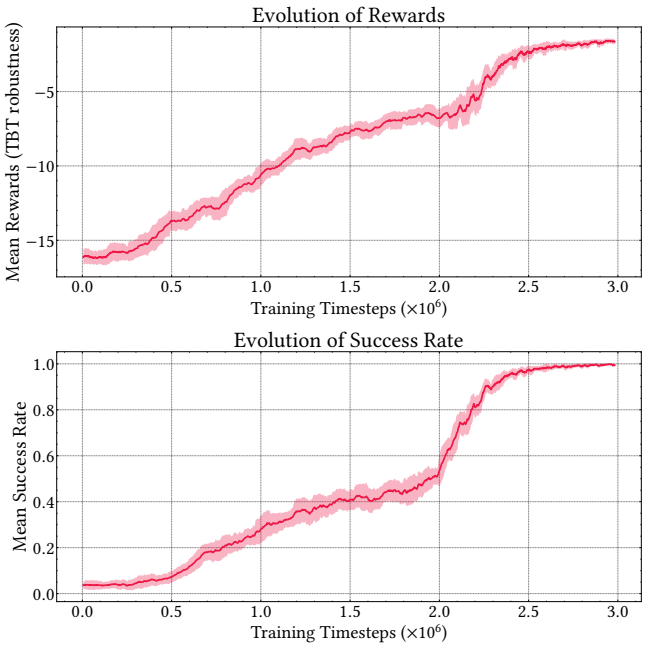


Figure 10: TQC training results for the pick-and-place task. Top: Rewards accumulated over the training session. Bottom: The progress of task success rates of the agent during training.

7 Conclusion

This paper discusses the need for formally reasoning about the increasingly popular robot programming paradigm of BTs. We

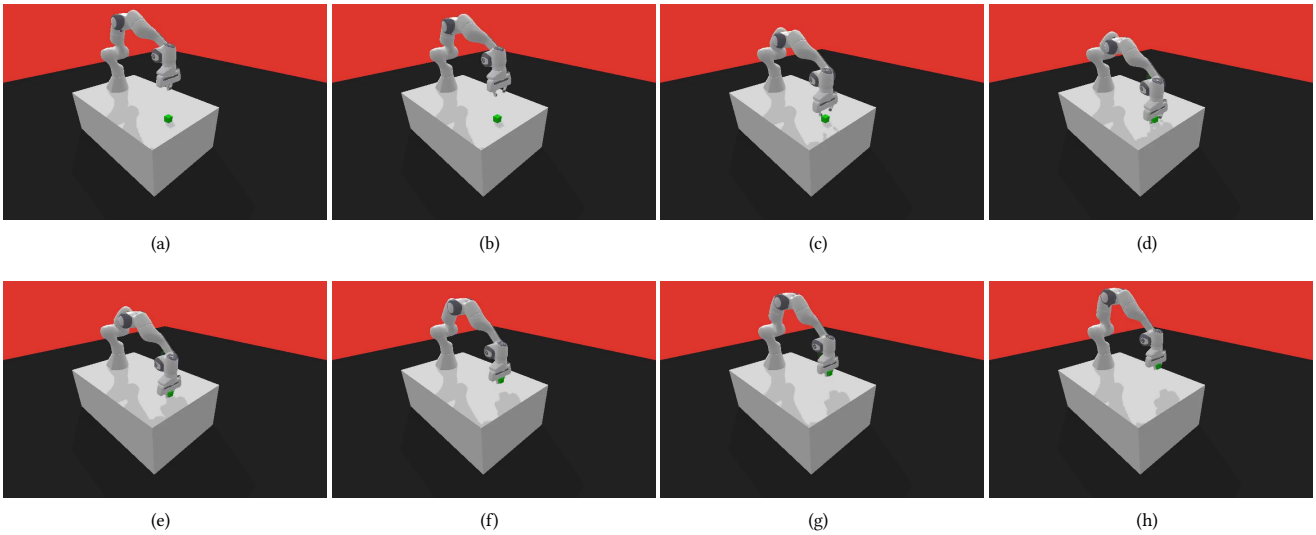


Figure 11: Sample trajectory from the control synthesized via automata-guided RL policy.

propose a novel framework for translating a BT to networks of TA with the introduction of hierarchical temporal logic for describing the structure of the BT. We define the qualitative semantics of this logic for monitoring the BT control and execution nodes, and show the language equivalence and bisimilarity between the BT and its corresponding timed automaton. The translated BT is used for falsification and control synthesis with Uppaal, and also used for control synthesis in robotic tasks involving safe navigation and pick-and-place manipulation. As part of future work, we will investigate dynamic generation of BTs and using BT-automaton-guided control synthesis for long-horizon and multi-agent tasks. BTs can address the inherent nature of myopic behaviors in deep RL that tend to result in neural network collapse for long-horizon tasks [40, 46]. The modularity of BTs can be further extended to enable skill-based RL [7].

Acknowledgments

This work was supported by the following grants: Office of Naval Research grant No. N000141712622; U.S. Army Contracting Command, Army Research Lab contract No. W911NF-23-2-0040; National Science Foundation grant No. 2422282; National Science Foundation grant No. 2219101 and Air Force Office of Scientific Research grant No. FA9550-23-1-0529.

References

[1] Mohammad Afzal, Sankalp Gambhir, Ashutosh Gupta, Krishna S, Ashutosh Trivedi, and Alvaro Velasquez. 2023. LTL-Based Non-Markovian Inverse Reinforcement Learning. In *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems* (London, United Kingdom) *(AAMAS '23)*. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 2857–2859.

[2] D. Aksaray, A. Jones, Z. Kong, M. Schwager, and C. Belta. 2016. Q-Learning for Robust Satisfaction of Signal Temporal Logic Specifications. In *2016 IEEE 55th Conference on Decision and Control (CDC)*. 6565–6570.

[3] S Akshay, Paul Gastin, R Govind, and B Srivathsan. 2024. MITL model checking via generalized timed automata and a new liveness algorithm. *arXiv preprint arXiv:2407.08452* (2024).

[4] Rajeev Alur and David L Dill. 1994. A theory of timed automata. *Theoretical computer science* 126, 2 (1994), 183–235.

[5] Rajeev Alur, Tomás Feder, and Thomas A Henzinger. 1996. The benefits of relaxing punctuality. *Journal of the ACM (JACM)* 43, 1 (1996), 116–146.

[6] Aaron D. Ames, Samuel Coogan, Magnus Egerstedt, Gennaro Notomista, Koushil Sreenath, and Paulo Tabuada. 2019. Control Barrier Functions: Theory and Applications. arXiv:1903.11199 [cs.SY] https://arxiv.org/abs/1903.11199

[7] Akhil Bagaria, Jason Senthil, Matthew Slivinski, and George Konidaris. 2021. Robustly Learning Composable Options in Deep Reinforcement Learning. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, Zhi-Hua Zhou (Ed.). International Joint Conferences on Artificial Intelligence Organization, 2161–2169. https://doi.org/10.24963/ijcai.2021/298 Main Track.

[8] Anand Balakrishnan and Jyotirmoy V. Deshmukh. 2019. Structured Reward Shaping using Signal Temporal Logic specifications. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2019, Macau, SAR, China, November 3-8, 2019*. IEEE, 3481–3486.

[9] Gerd Behrmann, Alexandre David, and Kim G Larsen. 2004. A tutorial on uppaal. *Formal methods for the design of real-time systems* (2004), 200–236.

[10] Mathieu Blondel, Quentin Berthet, Marco Cuturi, Roy Frostig, Stephan Hoyer, Felipe Llinares-López, Fabian Pedregosa, and Jean-Philippe Vert. 2021. Efficient and Modular Implicit Differentiation. *arXiv preprint arXiv:2105.15183* (2021).

[11] Thomas Brihaye, Gilles Geeraerts, Hsi-Ming Ho, and Benjamin Monmege. 2017. MightyL: A compositional translation from MITL to timed automata. In *International Conference on Computer Aided Verification*. Springer, 421–440.

[12] Alessandro Cimatti, Luca Geatti, Nicola Gigante, Angelo Montanari, and Stefano Tonetta. 2020. Reactive Synthesis from Extended Bounded Response LTL Specifications. In *2020 Formal Methods in Computer Aided Design, FMCAD 2020, Haifa, Israel, September 21-24, 2020*. IEEE, 83–92. https://doi.org/10.34727/2020/isbn.978-3-85448-042-6_15

[13] Michele Colledanchise, Richard M. Murray, and Petter Ögren. 2017. Synthesis of correct-by-construction behavior trees. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 6039–6046. https://doi.org/10.1109/IROS.2017.8206502

[14] Michele Colledanchise and Petter Ögren. 2014. How Behavior Trees modularize robustness and safety in hybrid systems. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 1482–1488. https://doi.org/10.1109/IROS.2014.6942752

[15] Michele Colledanchise and Petter Ögren. 2018. Behavior Trees in Robotics and AI. https://doi.org/10.1201/9780429489105

[16] Robert Colvin, Lars Grunske, and Kirsten Winter. 2008. Timed behavior trees for failure mode and effects analysis of time-critical systems. *Journal of Systems and Software* 81, 12 (2008), 2163–2182.

[17] Alessio De Luca, Luca Muratore, and Nikos G. Tsagarakis. 2023. Autonomous Navigation With Online Replanning and Recovery Behaviors for Wheeled-Legged Robots Using Behavior Trees. *IEEE Robotics and Automation Letters* 8, 10 (2023), 6803–6810. https://doi.org/10.1109/LRA.2023.3313052

[18] Alexandre Donzé and Oded Maler. 2010. Robust Satisfaction of Temporal Logic over Real-Valued Signals. In *FORMATS*.

[19] R Geoff Dromey. 2003. From requirements to design: Formalizing the key steps. In *First International Conference onSoftware Engineering and Formal Methods, 2003. Proceedings.* IEEE, 2–11.

[20] Davide Faconti, Michele Colledanchise, et al. 2024. BehaviorTree.CPP. https://github.com/BehaviorTree/BehaviorTree.CPP

[21] Georgios E. Fainekos and George J. Pappas. 2009. Robustness of temporal logic specifications for continuous-time signals. *Theoretical Computer Science* (2009).

[22] Quentin Gallouédec, Nicolas Cazin, Emmanuel Dellandréa, and Liming Chen. 2021. panda-gym: Open-Source Goal-Conditioned Environments for Robotic Learning. *4th Robot Learning Workshop: Self-Supervised and Lifelong Learning at NeurIPS* (2021).

[23] Lars Grunske, Kirsten Winter, and Robert Colvin. 2007. Timed behavior trees and their application to verifying real-time systems. In *2007 Australian Software Engineering Conference (ASWEC'07)*. IEEE, 211–222.

[24] M. Hendriks, Wang Yi, P. Petterson, J. Hakansson, K.G. Larsen, A. David, G. Behrmann, M. Hendriks, Wang Yi, P. Petterson, J. Hakansson, K.G. Larsen, A. David, and G. Behrmann. 2006. UPPAAL 4.0. In *Third International Conference on the Quantitative Evaluation of Systems - (QEST'06)*. 125–126. https://doi.org/10.1109/QEST.2006.59

[25] Craig Innes and Subramanian Ramamoorthy. 2020. Elaborating on Learned Demonstrations with Temporal Logic Specifications. In *Robotics: Science and Systems XVI, Virtual Event / Corvalis, Oregon, USA, July 12-16, 2020*, Marc Toussaint, Antonio Bicchi, and Tucker Hermans (Eds.). https://doi.org/10.15607/RSS.2020.XVI.004

[26] Yuqian Jiang, Suda Bharadwaj, Bo Wu, Rishi Shah, Ufuk Topcu, and Peter Stone. 2021. Temporal-Logic-Based Reward Shaping for Continuing Reinforcement Learning Tasks. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*. AAAI Press, 7995–8003. https://ojs.aaai.org/index.php/AAAI/article/view/16975

[27] Arsenii Kuznetsov, Pavel Shvechikov, Alexander Grishin, and Dmitry Vetrov. 2020. Controlling Overestimation Bias with Truncated Mixture of Continuous Distributional Quantile Critics. In *Proceedings of the 37th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 119)*, Hal Daumé III and Aarti Singh (Eds.). PMLR, 5556–5566. https://proceedings.mlr.press/v119/kuznetsov20a.html

[28] Xiao Li, Yao Ma, and Calin Belta. 2018. Automata Guided Reinforcement Learning With Demonstrations. *CoRR* abs/1809.06305 (2018). arXiv:1809.06305 http://arxiv.org/abs/1809.06305

[29] X. Li, C. Vasile, and C. Belta. 2017. Reinforcement Learning with Temporal Logic Rewards. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 3834–3839.

[30] Zhenyu Lin and John S Baras. 2020. Metric interval temporal logic based reinforcement learning with runtime monitoring and self-correction. In *2020 American Control Conference (ACC)*. IEEE, 5400–5406.

[31] Steve Macenski, Francisco Martín, Ruffin White, and Jonatan Ginés Clavero. 2020. The Marathon 2: A Navigation System. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. https://github.com/ros-planning/navigation2

[32] Oded Maler and Dejan Nickovic. 2004. Monitoring Temporal Properties of Continuous Signals. In *FORMATS*. Springer.

[33] Oded Maler, Dejan Nickovic, and Amir Pnueli. 2006. From MITL to timed automata. In *Formal Modeling and Analysis of Timed Systems: 4th International Conference, FORMATS 2006, Paris, France, September 25-27, 2006. Proceedings 4*. Springer, 274–289.

[34] Alejandro Marzinotto, Michele Colledanchise, Christian Smith, and Petter Ögren. 2014. Towards a unified behavior trees framework for robot control. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*. 5420–5427. https://doi.org/10.1109/ICRA.2014.6907656

[35] R. Milner. 1989. *Communication and concurrency*. Prentice-Hall, Inc., USA.

[36] Dejan Ničković and Tomoya Yamaguchi. 2020. RTAMT: Online robustness monitors from STL. In *International Symposium on Automated Technology for Verification and Analysis*. Springer, 564–571.

[37] Alexandros Nikou, Dimitris Boskos, Jana Tumova, and Dimos V Dimarogonas. 2018. On the timed temporal logic planning of coupled multi-agent systems. *Automatica* 97 (2018), 339–345.

[38] Amir Pnueli. 1977. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*. 46–57. https://doi.org/10.1109/SFCS.1977.32

[39] Aniruddh G. Puranic, Jyotirmoy V. Deshmukh, and Stefanos Nikolaidis. 2023. Signal Temporal Logic-Guided Apprenticeship Learning. arXiv:2311.05084 [cs.RO]

[40] Harish Ravichandar, Athanasios S. Polydoros, Sonia Chernova, and Aude Billard. 2020. Recent Advances in Robot Learning from Demonstration. *Annual Review of Control, Robotics, and Autonomous Systems* 3, 1 (2020), 297–330.

[41] Arash Saberi, Jan Groote, and Sarmen Keshishzadeh. 2013. Analysis of path planning algorithms: a formal verification-based approach. In *Artificial Life Conference Proceedings*. MIT Press One Rogers Street, Cambridge, MA 02142-1209, USA journals-info …, 232–239.

[42] Sebastian Schirmer, Jasdeep Singh, Emily Jensen, Johann Dauer, Bernd Finkbeiner, and Sriram Sankaranarayanan. 2024. Temporal Behavior Trees: Robustness and Segmentation. In *Proceedings of the 27th ACM International Conference on Hybrid Systems: Computation and Control* (Hong Kong SAR, China) *(HSCC '24)*. Association for Computing Machinery, New York, NY, USA, Article 9, 14 pages. https://doi.org/10.1145/3641513.3650180

[43] Serena Serafina Serbinowska and Taylor T. Johnson. 2022. BehaVerify: Verifying Temporal Logic Specifications for Behavior Trees. In *Software Engineering and Formal Methods*, Bernd-Holger Schlingloff and Ming Chai (Eds.). Springer International Publishing, Cham, 307–323.

[44] V. Sklyarov. 1999. Hierarchical finite-state machines and their use for digital control. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 7, 2 (1999), 222–228. https://doi.org/10.1109/92.766749

[45] Daniel Stonier et al. 2023. Py Trees. https://github.com/splintered-reality/py_trees

[46] Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction* (second ed.). The MIT Press, Cambridge, MA.

[47] Haotian Zhou, Yunhan Lin, Longwu Yan, Jihong Zhu, and Huasong Min. 2024. LLM-BT: Performing Robotic Adaptive Tasks based on Large Language Models and Behavior Trees. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*. 16655–16661. https://doi.org/10.1109/ICRA57147.2024.10610183

[48] Yuchen Zhou, Dipankar Maity, and John S Baras. 2016. Timed automata approach for motion planning using metric interval temporal logic. In *2016 European Control Conference (ECC)*. IEEE, 690–695.