Imitation Learning for Highlevel Robot Behavior in the Context of Elderly Assistance

Imitationslernen für Roboterverhalten im Kontext von Assistenzrobotern für das 3. und 4. Lebensalter

Bachelor-Thesis von Matthias Lang aus Rothenburg ob der Tauber

Tag der Einreichung: 15.10.2019

1. Gutachten: Prof. Dr. Jan Peters
2. Gutachten: Dorothea Koert

TECHNISCHE UNIVERSITÄT DARMSTADT

Imitation Learning for Highlevel Robot Behavior in the Context of Elderly Assistance
Imitationslernen für Roboterverhalten im Kontext von Assistenzrobotern
für das 3. und 4. Lebensalter

Vorgelegte Bachelor-Thesis von Matthias Lang aus Rothenburg ob der Tauber

1. Gutachten: Prof. Dr. Jan Peters
2. Gutachten: Dorothea Koert

Tag der Einreichung: 15.10.2019

Erklärung zur Abschlussarbeit gemäß § 22 Abs. 7 und § 23 Abs. 7 APB TU Darmstadt

Hiermit versichere ich, Matthias Lang, die vorliegende Bachelor-Thesis gemäß § 22 Abs. 7 APB der TU Darmstadt ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Falle eines Plagiats (§38 Abs.2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung gemäß § 23 Abs. 7 APB überein.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

Datum/Date	Unterschrift/Signature

Abstract

Robots could potentially support and assist elderly people in everyday situations. Furthermore, they could also reduce time constraints of caregivers in nursing homes. In contrast to classical factory settings, an assistive robot cannot be pre-programmed for every situation because of the diversity of possibly needed tasks and its varying environment. The robot's ability to learn new behaviors is crucial for the adaption to new environments and the people interacting with it. But the challenge to develop a robot which has to interact with many different people is also an advantage, in that the people surrounding it are experts in their field and can teach new behaviors directly. Caregivers and even elderly people could be enabled to teach the robot without the help of a programmer, by generating its high-level behavior from demonstrations. There are multiple ways of modeling the high-level behavior of a robot in a nursing environment. Behavior Trees (BTs) can model expressive and reactive behaviors while staying simple and understandable. But the automatic generation of BTs from demonstration is an area of active research. In this thesis, we propose three ways of generating BTs from demonstration data. We adapt the approach of Planning and Acting using Behavior Trees (PA-BT) in two different ways to be used with demonstration data and compare it to our own approach based on topological action ordering and BT specialization. Furthermore, a small study about teaching a robot is performed and a pruning algorithm is proposed. Experimental evaluations on a pick and place task show that usable BTs can be generated from demonstration and that the generated behaviors can reliably reproduce the demonstration data. However, pruning is necessary to improve readability.

Zusammenfassung

Roboter können ältere Menschen in täglichen Situation potenziell unterstützen und assistieren. Zudem könnten sie die Arbeitsbelastung von Pflegepersonal in Altenheimen reduzieren. Im Gegensatz zu einer klassischen Fabrik-Situation kann ein Assistenzroboter, aufgrund der Vielfalt an möglicherweise benötigten Aufgaben und seinem sich verändernden Umfeld, nicht für jede Situation vorprogrammiert werden. Die Fähigkeit, neue Verhalten zu lernen ist daher wichtig für die Anpassung des Roboters an neue Umgebungen und Menschen, die mit ihm interagieren. Die Herausforderung einen Roboter zu entwickeln, der mit vielen verschiedenen Personen interagieren muss, ist aber auch ein Vorteil, da die Menschen um ihn herum Expert*innen in ihrem Gebiet sind und ihm neue Verhalten direkt beibringen können. Pflegepersonal und auch Senior'innen könnte es ermöglicht werden, dem Roboter neue Verhalten ohne die Hilfe einer Programmiererin oder eines Programmierers, mittels Imitationslernen, beizubringen, indem das Verhalten aus Demonstrationen generiert wird. Es gibt vielfache Arten das Verhalten eines Roboters im Umfeld eines Altenheims zu modellieren. Behavior Trees (BTs) können expressive und reaktive Verhalten, einfach und verständlich, modellieren. Allerdings ist die automatische Generierung von BTs aus Demonstrationen ein aktives Forschungsfeld. In dieser Thesis schlagen wir drei Wege zur BT-Generierung aus Demonstrationen vor. Wir adaptieren den Ansatz Planning and Acting using Behavior Trees (PA-BT) in zwei verschiedenen Arten, um mit Demonstrationen zu arbeiten und vergleichen ihn mit unserem eigenen Ansatz, der auf der topologischen Sortierung von Aktionen und der Spezialisierung eines BTs basiert. Zudem führen wir eine Studie, wie Robotern neue Verhalten beigebracht werden können und schlagen einen Pruning Algorithmus vor. Die Evaluation unserer Experimente mit der Aufgabe, einen Ball zu nehmen und zu platzieren, zeigen, dass verwendbare BTs aus Demonstrationen generiert werden können und, dass die generierten BTs die Demonstrationsdaten reproduzieren können. Allerdings ist Pruning nötig, um die Lesbarkeit zu steigern.

Acknowledgments

While working for this thesis, I was accompanied by many wonderful and inspiring people. First of all, I would like to thank my supervisor, Dorothea Koert, for her individual and very encouraging support. I would like to thank Prof. Jan Peters for giving me the opportunity to work for the Intelligent Autonomous Systems (IAS) group. Furthermore, I thank the participants of the performed study. Lastly, I would like to thank my wife, my family, my friends and my colleagues for making this thesis possible.

Contents

Figures and Tables

List of Figures

List of Tables

List of Algorithms

Abbreviations, Symbols and Operators

List of Abbreviations

Notation	Description
ABL	A Behavior Language
BT	Behavior Tree
CSV	Comma-Seperated Values
FSM	Finite State Machine
GUI	Graphical User Interface
IAS	Intelligent Autonomous Systems
KoBo	Cooperative assistive robot (from German: **Ko**operativer Assistenzro**bo**ter)
LfD	Learning from Demonstration
LTL	Linear Temporal Logic
LWPR	Locally Weighted Projection Regression
NPC	Non-Player character
PA-BT	Planning and Acting using Behavior Trees
ROS	Robot Operating System
TBT	Trained Behavior Trees
XML	Extensible Markup Language

List of Symbols

Notation	Description
A	finite set of pre-defined Actions
a	one action
C	finite set of pre-defined Conditions
Δ	set of transitions
e	one external action
s	state, represented by Conditions
δ	one transition, consisting either of initial state, action, state after the action (s, a, s') or initial state, external action, state after the external action (s, e, s')

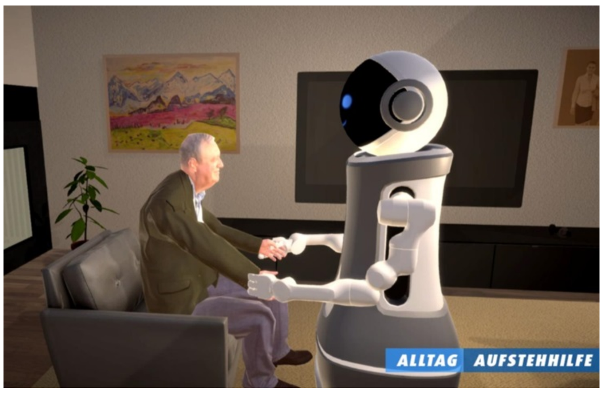
List of Operators

Notation	Description	Operator
childOf	The child node of a Behavior Tree node n	childOf(n)
$\neg$	The logical negation of Condition C	$\neg C$
parentOf	The parent node of a Behavior Tree node n	parentOf(n)
typeOf	The type of a Behavior Tree node n	typeOf(n)

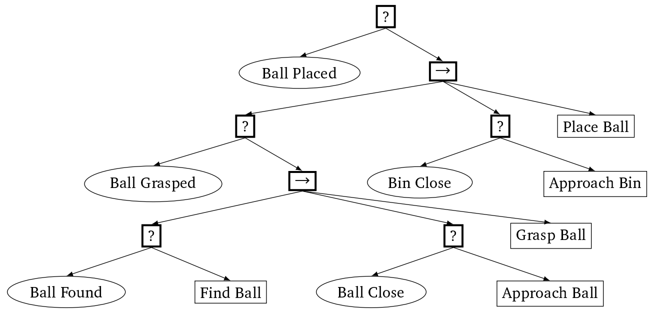
1 Introduction

1.1 Motivation

In 2018, according to the *Bundesagentur für Arbeit* in Germany, on average 23,900 job offers for elderly care plus an additional 15,700 nursing positions could not be filled [Bundesagentur für Arbeit, 2019, p. 12]. In addition to this development, an aging effect will change the German demographic profoundly [Allmendinger and Ebner, 2006, p. 227], leading to even more seniors in need of assistance and less young people to fill those open positions. Furthermore, these problems are no individual challenge of Germany, but for many other countries in Europe [Doblhammer and Ziegler, 2006, p. 267] as well. The goal of the KoBo34[1] project is to fill this gap and to develop a humanoid robot (see figure 1.1 (a)), which can assist elderly people in retirement homes. It is a cooperation of *Technische Universität Darmstadt*, *Technical University of Munich* and *Rosenheim University of Applied Sciences* with the *Bundesministerium für Bildung und Forschung* and has a timeframe of 3 years, starting in July 2018. The project takes place in close cooperation with the robot manufacturer *FRANKA EMIKA GmbH* and the nursing home *Alten- und Pflegeheim Lenzheim Garmisch-Partenkirchen*.



(a)



(b)

Figure 1.1.: Figure (a) shows, an official KoBo34 illustration[2], depicting the Kobo robot interacting with an elderly person and helping him to stand up. The project's goal is to develop a humanoid robot, which can assist elderly people in retirement homes. Figure (b) shows an example Behavior Tree, modeling a behavior for a robot to pick up a ball. Behavior Trees originally have been developed for modeling behavior of autonomous agents in video games and are, besides Finite State Machines, a possible way of representing a robot's behavior.

The KoBo34 project identified multiple use cases, in which the robot could assist caregivers and elderly people. In all these use cases, the robot has to interact with people in a varying environment. Therefore, in contrast to a robot in a controlled environment, like for example a classical factory setting, it has to be able to change its current behavior and to learn new skills. Furthermore, it is of significant importance that its high-level behavior is comprehensible to the people around him and adaptable to its environment. To achieve this, it is important to find an intuitive way of interaction with the caregivers and elderly people, to enable them to teach the robot without a programmer or a trained specialist. One possible solution could be imitation learning, also called *Learning from Demonstration* (LfD), where a policy is learned from examples, provided by a teacher [Argall et al., 2009]. These examples, or demonstrations, are recorded as state-action pairs and then used to reproduce the demonstrated behavior. If possible, the algorithm can even learn *online*, i.e. adapt to new demonstrations, without reasoning over the entire dataset. If an online approach can be implemented, it would be possible for the teacher, in this case a caregiver or elderly person, to simply correct the robot in its behavior and to witness an immediate change. Finally, the structure of the behavior has to be easily understandable. In case, a generated behavior has to be understood in detail, it cannot be overly complicated or even a black box, like for

[1] The name KoBo34 is a combination of the german words "**Ko**operativer Assistenzro**bo**ter", with the inidices 3 and 4, standing for the third and fourth period of life. The third period of life usually starts with retirement around 65 years and stands for a more active phase, whereas the fourth period usually is characterized by very old age and the need for medical assistance [Dorsch - Lexikon der Psychologie, 2019].

[2] Image taken from: `https://www.technik-zum-menschen-bringen.de/projekte/kobo34`. Last accessed on Sep 23, 2019.

example neural nets [Adadi and Berrada, 2018]. For the use in a nursing environment, the executed behavior has to be deterministic and traceable.
To meet these and other prerequisites set by the KoBo34 project, a suitable high-level behavior representation has to be found. There are multiple ways of modeling behavior with different advantages and possibilities. Therefore, one part of this thesis is to show and compare these advantages and differences. Two major behavioral structures and their capabilites are compared: Finite State Machines (FSMs), which model a behavior with individual states and transitions in between them and Behavior Trees (BTs), which are single-rooted, directed trees, which send a *tick* signal from the root node over control flow nodes to its leaf nodes that are then executed (see figure 1.1 (b)). BTs originally have been developed for the use in artificial intelligence for computer games [Isla, 2005]. They are mainly used to increase modularity, because "individual behaviors can easily be reused in the context of another higher-level behavior, without needing to specify how they relate to subsequent behaviors" [Bagnell et al., 2012, p. 3] and only recently received more attention in academia [Colledanchise and Ögren, 2017, p. 4]. Another key advantage of BTs, being a visual behavior representation, is the possibility for non-programmers to understand and create behaviors, as expressive as traditionally-written programs [Paxton et al., 2017, p. 4].
This thesis, therefore tries to evaluate if and how BTs could be used for the KoBo34 project and what kind of advantages they could bring to the project instead of FSMs. Furthermore, different ways of automatically creating BTs by Learning from Demonstration are evaluated. This is an active field of research [Colledanchise and Ögren, 2017, pp. 153-154], because LfD is not very common for the games industry (i.e. controlling a non-player character (NPC) to teach the desired behavior). Additionally, a way to create the BTs, as human readable as possible, has to be found. As Colledanchise et al. state: "Unfortunately, using learning from demonstration approaches the learned BT easily becomes very large, as each trace is directly mapped into a new sub-BT" [Colledanchise and Ögren, 2017, p. 154]. To increase readability of the generated BTs, redundant nodes have to be identified and pruned.

1.2 Structure of this Thesis

This thesis is structured in the following chapters:

Chapter 2 summarizes fundamental information about Finite State Machines, Behavior Trees and the automatic expansion of these behavioral representations. Furthermore, an overview over the related work to automatic expansion of FSMs and BTs is given.

Chapter 3 defines the problem statement and compares the capabilities of FSMs and BTs on the basis of 4 use cases. Finally, it provides the rationale for the use of BTs in further work.

Chapter 4 presents three different approaches for the automatic expansion of Behavior Trees, namely *Planning and Acting using Behavior Trees* (PA-BT) with given actions and goal conditions, PA-BT with automatic action and goal state detection from demonstration data and automatic action sequence creation and BT specialization on demonstration data.

Chapter 5 presents the implementation of the BT framework and the setup for the three approaches for automatic behavior generation. Additionally, a short manual for the implemented Graphical User Interface is given.

Chapter 6 evaluates the results of a study taken to find out how robot teaching could be performed with elderly people and shows the experimental results of the three implemented approaches.

Chapter 7 resumes the outcome and discusses possibilities for future work in the automatic expansion of Behavior Trees via imitation learning.

The **Appendix** contains the questionnaire and the results of the performed study about ways of robot teaching.

2 Foundations & Related Work

In this chapter, the fundamental concepts of Finite State Machines (FSMs) [Hopcroft et al., 2014] and Behavior Trees (BTs) [Colledanchise and Ögren, 2017] will be introduced. These ways of representing behaviors can be used to model robot behavior. Lastly, an overview is given over the related work to the direct automatic expansion of FSMs and BTs and possible intermediate structures to generate FSMs or BTs from.

2.1 Finite State Machines (FSMs)

In this section, the formal defintion of *Finite State Machines* (FSMs) is given and the extension to Hierarchical Finite State Machines is discussed.

2.1.1 Basic Finite State Machines

A FSM is an abstract machine that is defined by a finite set of states S, an initial state $s_0 \in S$, a finite set of possible inputs Σ and a set of transitions Δ between these states. Each transition is defined by its origin state $s \in S$, the input for this transition $i \in \Sigma$ and the final state after this transition $s' \in S$. The abstract machine can always only be in exactly one state at any given time [Hopcroft et al., 2014]. FSMs can be shown either in a transition matrix (see table 2.1) or a state transition diagram (see figure 2.1) [Wagner et al., 2006].

To / *From*	s	s’
s	-	i
s’	-	-

Table 2.1.: A transition matrix of a simple Finite State Machine with start state s and a transition to a second state s' with input i. If the current state of the FSM is state s and the input is equal to i, the state changes to state s'. A state transition diagram of this FSM can be seen in figure 2.1.

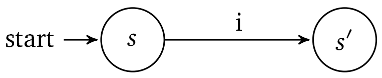


Figure 2.1.: A state transition diagram of the same simple Finite State Machine as seen in table 2.1 with start state s and a transition to a second state s' with input i. If the current state of the FSM is state s and the input is equal to i, the state changes to state s'.

There are two types of Finite State Machines: deterministic finite automaton (DFA) and nondeterministic finite automaton (NFA), seen in figure 2.2 and figure 2.3. A DFA has always at most one possible move per input i in state s. In contrast to the deterministic behavior of a DFA, a NFA can have several choices for the next move [Hopcroft et al., 2014].
Since deterministic behavior is a requirement for the robot behavior in this thesis, we will concentrate only on DFAs.

2.1.2 Hierarchical Finite State Machines

The concept of hierarchical Finite State Machines, introduced by David Harel in 1987, extends the concept of FSMs by adding hierarchical levels [Harel, 1987]. Each state in the top-level FSM can be represented by another lower-level FSM, as seen in figure 2.4. While the state of the top-level FSM does not change, the internal state refinement of the lower-level FSM can still change. These lower levels can have a history, i.e. when the top-level FSM leaves this state and comes

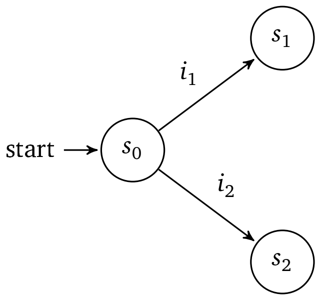


Figure 2.2.: A deterministic finite automaton (DFA). The DFA has in every state at most one possible transition per input. In contrast to the NFA seen in figure 2.3, the state transitions of this abstract machine are deterministic.

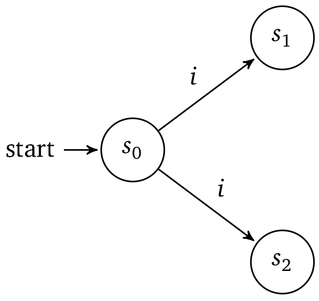


Figure 2.3.: A nondeterministic finite automaton (NFA). In contrast to the DFA seen in figure 2.3, this NFA has multiple possible transitions in state s_0 for the input i. Therefore after receiving input i in state s_0 the FSM could be in state s_1 or s_2. So the behavior of this FSM is nondeterministic.

back later, the internal state will be retained. Without a history, the lower-level FSMs always restart at the same internal start state. Fundamentally, the added hierarchy does not reduce the number of states and adds no further computational capability. In other words, they are equivalent and a transfer between them is possible. But it makes the FSM easier to read and more intuitive [Girault et al., 1999]. Furthermore, when used for the modelling of robot behavior, already existing sub-behaviors can be reused, which increases the modularity of FSMs.

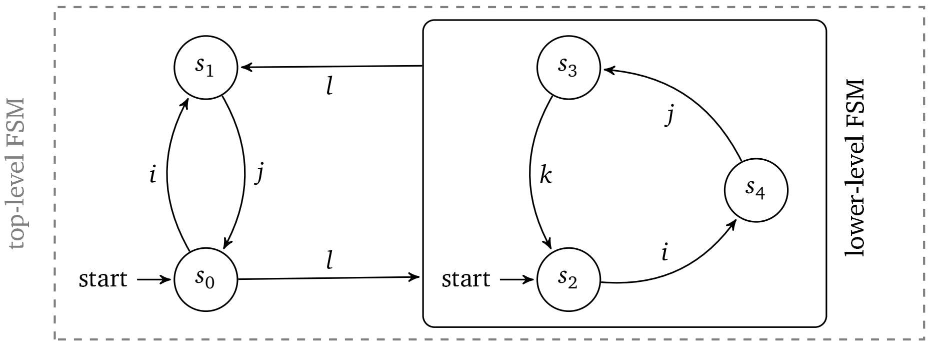


Figure 2.4.: An example hierarchical FSM. The state of the top-level FSM does not change, when the lower level FSM consisting of s_2, s_3 and s_4, is executed. The lower-level FSM can always start in the same state (in this case s_2) when reached or retain the internal state and resume in the previous state when reached again.

2.2 Behavior Trees (BTs)

Behavior Trees (BTs) have mainly been developed for autonomous agents in computer games [Isla, 2005]. BTs represent the execution of actions based on conditions and observations in a system via a graphic modeling language [Colledanchise et al., 2016]. There are some key qualities which distinguish them from the use of finite state machines to switch in between tasks. BTs were developed to increase *modularity* and enable furthermore *reactive* behavior. Reactiveness is the ability to easily react to (unforeseen) changes, for example a robot dropping a ball [Colledanchise and Ögren, 2017].
A BT is a directed rooted tree. The leaf nodes are called *execution nodes*, while all other nodes are called *control flow nodes*. There are two types of execution nodes (Action and Condition), and four categories of control flow nodes (Se-

quence, Fallback, Parallel, and Decorator). An overview of all node types and their return values can be seen in table 2.2 which will be explained in more detail further below. The execution of a BT starts at the root node, which generates signals called *ticks* with a given frequency. These ticks are sent to its children, which are executed only if they receive a tick. Every executing child immediately returns either `RUNNING` if it is still executing, `SUCCESS` if its goal has been achieved or `FAILURE`.

Node type	**Symbol**	**Succeeds**	**Fails**	**RUNNING**
Fallback	$\boxed{?}$	If one child succeeds	If all children fail	If one child returns Running
Sequence	$\boxed{\rightarrow}$	If all children succeed	If one child fails	If one child returns Running
Parallel	$\boxed{\rightrightarrows}$	If $\geq M$ children succeed	If $>N$ - M children fail	else
Action	$\boxed{\text{text}}$	Upon completion	If impossible to complete	During completion
Condition	(text)	If true	If false	Never
Decorator	◇	Custom	Custom	Custom

Table 2.2.: The node types of a BT [Colledanchise and Ögren, 2017] and their return values. When ticked, each node immediately returns either `SUCCESS`, `FAILURE` or `RUNNING`. *Fallback*, *Sequence*, *Parallel* and *Decorator* nodes are called control flow nodes and forward the tick signal to their children. *Action* and *Condition* nodes are called execution nodes and represent the leaf nodes of the BT.

The **Fallback** node routes the ticks to its children from left to right. When a child returns `FAILURE`, the tick is routed to the next child to the right. While a child returns `RUNNING` the Fallback node returns `RUNNING` as well. Only if all children return `FAILURE` the Fallback node returns `FAILURE` as well.

The **Sequence** node also routes the ticks to its children from left to right. When a child returns `SUCCESS`, the tick is routed to the next child to the right. While a child returns `RUNNING` the Sequence node returns `RUNNING` as well. Only if all children return `SUCCESS` the Sequence node returns `SUCCESS` as well.

The **Parallel** node routes the ticks to all its children at the same time. When M children return `SUCCESS`, the Parallel node returns `SUCCESS` as well. When $N - M + 1$ children return `FAILURE` (i.e. it is impossible that M children return `SUCCESS` afterwards), the Parallel node returns `FAILURE` as well. In all other cases, it returns `RUNNING`.

The **Action** node executes a command. It returns `RUNNING` while executing, `FAILURE` if the execution failed and `SUCCESS` if the execution succeeded.

The **Condition** node checks a proposition and returns `FAILURE` and `SUCCESS` accordingly. It never returns `RUNNING`.

The **Decorator** node can be used to manipulate the return of its single child. For example the return value of the child could be inverted, a maximum number of ticks or a maximum execution time could be defined.

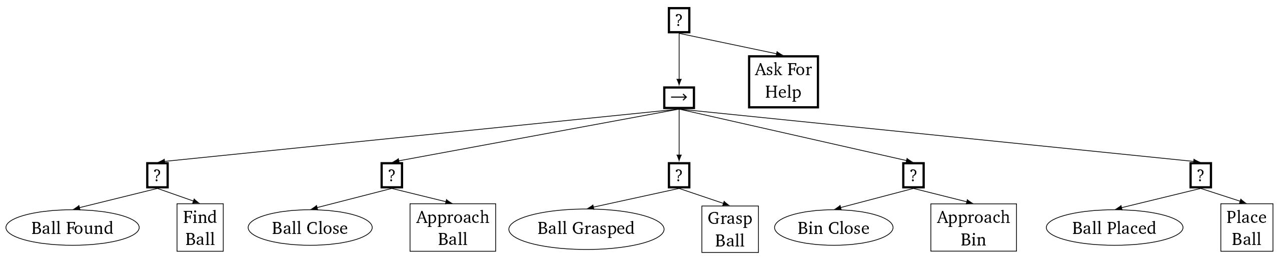


Figure 2.5.: BT encoding the behavior of a simple example [Colledanchise and Ögren, 2017] where a robot finds a ball, picks it up and puts it in a bin. The tick signal originates from the root node (in this case a Fallback node) and is forwarded to its children from left to right. In this case, the tick signal is sent from the root node, over a Sequence node, another Fallback node and to the Condition node *Ball Found*. If the robot has not already found the ball, this Condition node returns `FAILURE` and the tick signal is forwarded to the Action node *Find Ball*. If this Action node returns `SUCCESS`, the sequence is continued and the tick signal is sent to the next Condition node *Ball Close*. If the Action node returns `RUNNING`, the whole BT returns `RUNNING` and the execution of this tick cycle is finished. If the Action node returns `FAILURE`, the whole sequence fails and the root node, being a Fallback node, routes the tick signal to his next child, the Action node *Ask For Help*.

The example shown in figure 2.5 shows the task of finding a ball, approaching and grasping it, before approaching a bin to place the ball in it. The sequential behavior is organized via a Sequence node. The root node is a Fallback node, which routes the ticks to its children from left to right. Therefore, the Sequence node is ticked until it returns SUCCESS or FAILURE. In the case of a failure of the sequence, the Action node *Ask For Help* is ticked. The sequence of steps which have to be performed is organised with Fallback nodes containing pairs of a Condition node and an Action node. Therefore, the condition is either true or the appropriate action will be performed. In this case, the Action node never returns SUCCESS but changes the paired condition. This avoids the re-execution of an Action, once it has been executed. Another way to achieve this, is to use nodes with memory, which are indicated by the addition of a "*" symbol (see figure 2.6). Once a node has returned FAILURE or SUCCESS, control flow nodes with memory remember this and do not re-execute the action. Since this behavior can be achieved with normal control flow nodes as well, as seen in figure 2.7, nodes with memory can be seen as syntactic sugar.

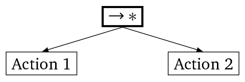


Figure 2.6.: Sequence composition with memory. Control flow nodes with memory are indicated by the symbol "*". A control flow node with memory does not re-execute all children to the left of the running child. Instead, it saves the return values of its children until the control flow node finishes in either SUCCESS or FAILURE.

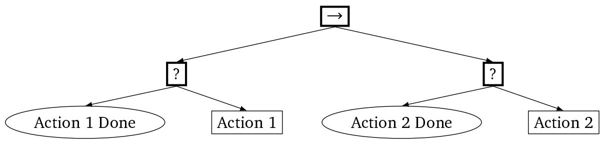


Figure 2.7.: BT that emulates the behavior of figure 2.6 using nodes without memory. If *Action 1* has been executed, the Condition node *Action 1 Done* changes its return value from FAILURE to SUCCESS. In this case, *Action 1* is not executed anymore, because of the Fallback node *Action 1 Done* and *Action 1* are connected to.

One advantage of Behavior Trees can be seen in the simple example in figure 2.5. If the Ball is already grasped, the Sequence node would tick the next Fallback node to approach the bin. If in this case, the robot drops the ball, *Ball Grasped* would change to FAILURE and the Behavior Tree would jump back to *Grasp Ball*. If in this case, the ball is not close anymore or even out of sight, the Behavior Tree could even jump back to *Find Ball*. This demonstrates the reactiveness of Behavior Trees.

Of course, we can only give a short introduction on Behavior Trees as part of this thesis. For further information consult for example *Behavior Trees in Robotics and AI* by Colledanchise and Ögren [Colledanchise and Ögren, 2017].

2.3 Related Work

In this section we briefly summarize the related work to the automatic expansion of FSMs and BTs. Additionally, related work where FSMs or BTs are expanded via a different intermediate behavioral representation, is discussed.

2.3.1 Automatic Expansion of FSMs

Grollman and Jenkins propose an approach for regression-based Robot Learning from Demonstration for use in a FSM controller [Grollman and Jenkins, 2010a]. They conclude, that additional information about a task, such as a segmentation of the demonstration data or the number of subtasks has to be given to fully learn a FSM. To achieve this, the number of subtasks has to be learned, their individual policies and the transitions between them [Grollman and Jenkins, 2010b]. They propose a learning approach based on Locally Weighted Projection Regression (LWPR) [Grollman and Jenkins, 2008]. They conclude, that learning a general FSM from demonstration is still an open problem. If the demonstration consists of two or more subtasks, perceptual aliasing (i.e. perceptions which are similar, but demand different actions) can occur.

Niekum at al. are learning a finite-state representation from unstructured demonstrations [Niekum et al., 2015]. They are applying Bayesian non-parametric methods to automatically detect repeated structure represented in the demonstration and to segment it into known skills. However, this work handles no high-level behavior, which we will try to model in this thesis.

Some approaches to expand BTs automatically already exist, for example described by Colledanchise et al. [Colledanchise et al., 2016, Colledanchise and Ögren, 2017]. The proposed method *Planning and Acting using Behavior Trees* is based on the idea of backchaining and the creation of atomic BTs, which are created to satisfy a specific condition as seen in figure 2.8. The expansion starts with a BT, only consisting of the goal conditions, which is then executed. If all conditions are met, the BT is successfully created. If a condition fails, it is replaced by the atomic BT which satisfies the failing condition. Then the BT is executed again multiple times and all failing conditions are replaced accordingly until the BT succeeds. To avoid an infinite number of expansions, every condition is expanded only once. This approach works only if the BT can be executed multiple times, i.e. there is an existing simulation, in which every action has specified pre- and post-conditions. Our approach should work on demonstration data instead of a simulation and running the behavior on the robot multiple times is not a viable option. Furthermore, the goal conditions have to be specified for this approach. Our own approach should be able to determine the goal conditions independently. Our own approach is actually based on PA-BT and will be discussed in further detail in section 4.1 and section 4.2.

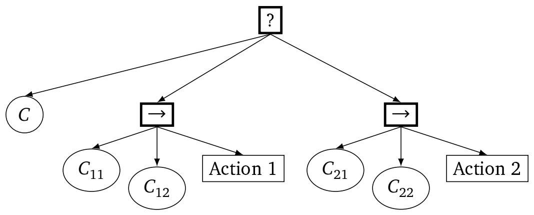


Figure 2.8.: General format of an atomic BT. The postcondition C can be achieved by multiple Actions with multiple individual preconditions. In this case, the return value of C can be changed by executing either Action 1 or Action 2. In this example, each Action has two preconditions (for example: C_{11} and C_{12} are preconditions for Action 1). In case, the postcondition C fails while execution, it is replaced with this atomic BT. Therefore, after this replacement, either Action 1 or Action 2 (depending on their preconditions) can change the return value of the postcondition C.

Sagredo-Olivenza et al. propose the concept of Trained Behavior Trees (TBTs) [Sagredo-Olivenza et al., 2017]. A minimal BT which contains a special trainer node, which is executed and trained by a designer. After the training, a decision tree is created using the C4.5 algorithm [Quinlan, 2014]. This decision tree is converted into rule clauses, which are then simplified. A BT with one parallel node is created which has all tasks as children. Each task is connected to the parallel node via a guard node which ensures the execution of the specific task according to the conditions learned in the decision tree (as seen in figure 2.9).

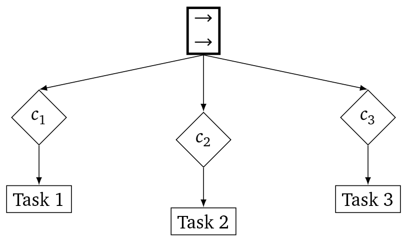


Figure 2.9.: Example of a Trained Behavior Tree created from a decision tree, which has been converted to rule clauses with conditions c_1, c_2 and c_3. Guard nodes with the corresponding conditions ensure the execution of the correct task, only when the corresponding condition is true. The root node is a Parallel node, sending the tick signal to all its children. Therefore, it is even possible that all Tasks are executed at the same time.

One advantage of this approach is that the BT does not have to be executed multiple times for its creation. Unfortunately the expansion using one parallel node and multiple guard nodes does not take into account all the possibilities of BTs. It is rather another representation of the created rule clauses.
Another approach, using motif-finding techniques from computational biology is proposed by Robertson and Watson [Robertson and Watson, 2015], using the real-time strategy game *StarCraft*. The goal is to find recurring action sequences in a BT to reduce its size. Therefore a maximally specific BT, which is clearly over-fit to the demonstration data, is created by adding each demonstration as an independent sequence as child of a selector node. This selector node can choose one child randomly or according to the current state. Common patterns of actions are then found and combined iteratively, using the *Gapped Local Alignment of Motifs* (GLAM2) software. A score for the similarity of two sequences is calculated and every match with a score over a defined threshold is combined to a new sequence, until no new common patterns

are found. This also generalizes the BT. With this approach, it was possible to reduce the number of nodes in an example BT from 218,832 to 71,294 nodes. Unfortunately, the performance of the newly created tree has not been tested and compared to the original one. This solution seems to work well with very big BTs and extensive demonstration material. In contrast, our approach has to deal with a small amount of demonstration data and create small, human-readable and easily understandable BTs.
Schwab and Hlavacs also propose to identify characteristic sequences [Schwab and Hlavacs, 2015], i.e. (sub-)sequences that most likely lead to the predefined goal. To achieve this, the state space is explored via Monte-Carlo Simulation to find action sequences leading to the goal state. Therefore, a decision-making algorithm $f(s)$, which returns an action sequence leading to the goal state from state s has to exist and therefore every action with its preconditions and effects on the state have to be known. Via a n-gram model, built from the created action sequences, the most characteristic action sequences to achieve the goal state are identified. These are then merged into a BT using evolutionary optimization. A population of BTs is initialized by randomly combining characteristic sequences which are then crossed over with other BTs and mutated. To find the best BT, the individual behavior is compared to the decision-making algorithm $f(s)$. This approach could be adapted to the problem of this thesis. Monte-Carlo Simulation would be replaced by demonstration and the decision-making algorithm $f(s)$ could be a function which returns known action sequences from the recorded demonstration. Therefore the state space is only represented by states, which were recorded while demonstrating. Unfortunately, the behavior structure should be computable within reasonable time and computational effort as defined in the prerequisites. An evolutionary approach could lead to a too high computational load.

2.3.3 FSM or BT Expansion from Other Behavioral Representations

There is also the possibility of synthesizing a FSM or a BT from another intermediate representation, which could be synthesized using the demonstration.
One possible intermediate could be the reactive planning language **A** **B**ehavior **L**anguage (ABL, pronounced "able") proposed by Weber et al. [Weber et al., 2010, Weber et al., 2011]. An algorithm to create a BT from ABL is even proposed by Colledanchise and Ögren [Colledanchise and Ögren, 2017]. However, the BT would only be used as an execution tool for ABL. The ABL structures can return either `SUCCESS` or `FAILURE`, and not `RUNNING` like BT nodes. Also, hand-made structures have to be used, which makes an automatic creation more difficult. Therefore, ABL structures are not reactive [Colledanchise et al., 2016, p. 2], which is one of the prerequisites of this thesis.
Another behavior representation to expand from could be linear temporal logic (LTL). Kress-Gazit et al. are proposing a way to synthesize a FSM from LTL formula via a game between the FSM and the environment [Kress-Gazit et al., 2011], based on the algorithm proposed by Piterman et al. [Piterman et al., 2006]. Only if the FSM wins no matter what the environment does, the FSM is finally generated. To deal with the state explosion problem, the FSM is created by implementing a receding horizon framework. Therefore, only states within reach of the current state are part of the computation. An open-source python-based implementation is provided. Unfortunately, the LTL formula are seen as given and there is no mentioning of how to learn these from demonstration, which is a prerequisite of this thesis.
The automatic generation of a FSM from user input and a description of the system is proposed by Maniatopoulos et al. [Maniatopoulos et al., 2016], using the same game algorithm proposed by Piterman et al. [Piterman et al., 2006]. To achieve this, the system's capabilities and constraints are modeled in LTL. A non-expert user then only has to specify a high-level task to generate the FSM which is based on the *FlexBE* GUI and the *Robot Operating System* (ROS). This task-oriented approach differs from our approach, which is based on an unknown system and learning from demonstration.

3 Comparison of the Use of Finite State Machines and Behavior Trees for Highlevel Robot Behavior

In this chapter, the problem statement is defined, the use cases defined by the Kobo34 project are described and the rationale for using Behavior Trees is discussed on the basis of the given use cases.

3.1 Problem Statement

The KoBo robot has a finite set of pre-defined, deterministic actions A and a finite set of conditions C which represent the state s. For KoBo to learn new behavior, it should be able to learn from demonstration. The demonstration is represented by a set of transitions Δ which either consist of an initial state s, an action a and the state s' after the action took place.

$$\delta = (s, a, s')$$

Or, for state changes without an active action which indicate external factors, the transition consists of an initial state s, an external action e and the state s'.

$$\delta = (s, e, s')$$

Our problem can be modeled as a set of actions A, a set of conditions C, a set of transitions Δ.

$$\theta = (A, C, \Delta)$$

The challenge is to automatically generate a behavior structure which reacts to the given states according to the given demonstration. It should also meet the following prerequisites:

- The behavior structure should be able to work with states stored in demonstration data.
- It should be able to react to inputs and to create outputs.
- It should react to unforeseen external changes as well.
- Behavioral changes at run-time should be possible.
- The behavior has to be deterministic.
- The representation should be easily understandable (i.e. human-readable).
- It should be expressive (i.e. able to express all needed behavior).
- A stable implementation should be offered.
- The structure should be generatable automatically.
- The automatic generation should be computable within reasonable time and computational effort.
- Extensibility and modularity should be enabled.

If the demonstration is not sufficient to generate a usable behavior structure, one possible solution could be to ask the demonstrator meaningful questions. But it is important to note that the amount of questions should be kept to an absolute minimum.

3.2 Use Cases

The KoBo34 project has defined four use cases for the robot (Kobo) in cooperation with the *Alten- und Pflegeheim Lenzheim Garmisch-Partenkirchen*[1]. In the following section we will describe the different Use Cases and their challenges. Use Case 2 is described in more detail to show the main differences of Behavior Trees and Finite State Machines.

[1] `https://altenhilfe.rummelsberger-diakonie.de/standorte/garmisch-partenkirchen-lenzheim/ueber-uns/`

3.2.1 Use Case 1: Dinner Scenario

One of the most requested tasks by caregivers is for Kobo to help during dinner. Some elderly people have problems to take their own food to the tables. The robot could present the menu and ask individuals for their choice, which it will then order in the kitchen. Finally, it should remember which person ordered which menu item and bring the prepared dinner trays to the tables in the common room and fetch used ones when the seniors have finished eating (see also illustration in figure 3.1). This would give the caregivers the possibility to spend more time with the seniors at the table instead of having to serve dinner for everyone.

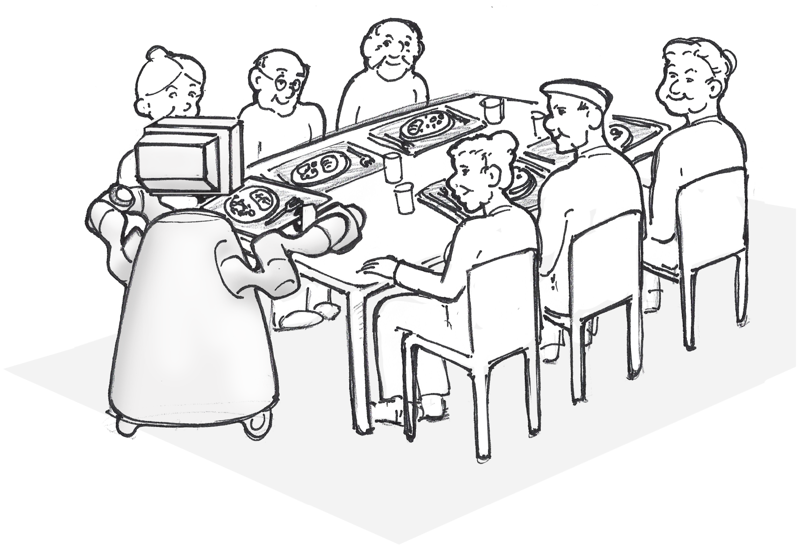


Figure 3.1.: Official KoBo34 illustration for Use Case 1: Kobo helping at lunch. The robot is presenting the menu, takes orders, brings prepared dinner trays to and removes used ones from the table.

3.2.2 Use Case 2: Offering Snacks and Drinks

The second use case demands much interaction with the elderly people. Kobo has to offer snacks and beverages to the seniors in between lunch hours. Its behavior should encourage to drink more and to stay hydrated (see also illustration in figure 3.2).

The Behavior Tree seen in figure 3.3 is modeled after the given KoBo use case. The whole behavior is organized as a sequence. The robot has to be requested by the caregiver to offer snacks or drinks. If it is not ordered to do so, it will simply wait for a request. Once requested, it starts by taking a tray and instructions. If his tray is not empty, the Fallback node will continue with another sequence where the robot finds an interaction and then offers snacks. One advantage of Behavior Trees can be seen in the fact, that even in case someone took the last snack without the robot recognizing the interaction and offering it, the Behavior Tree will fall back to another sequence which is triggered by *Tray is empty*. In this case, the robot will automatically return to the caregiver office. In the unforeseen event of the robot losing the tray, it will automatically return to *Take tray from human*.

This behavior can also be modeled as a FSM (as seen in figure 3.4). The individual actions are executed in the corresponding states.

This simple attempt to create a FSM with equal abilities lacks the reactiveness of the BT in figure 3.3. For example, to add the ability to end the request at any given state and to send the robot back to *Wait for request* a new transition from every other state back to the start state has to be created. The resulting more complex FSM, seen in figure 3.5, has four more transitions with the same corresponding input *terminate request*.

In this scenario the robot could also lose the interaction with a human while offering snacks or drinks. In this case the Behavior Tree would simply go back to *Find interaction with human*. If the FSM should cover this case as well, another edge has to be added. To exactly replicate the reactive behavior modeled in figure 3.3, several new transitions would have to be added.

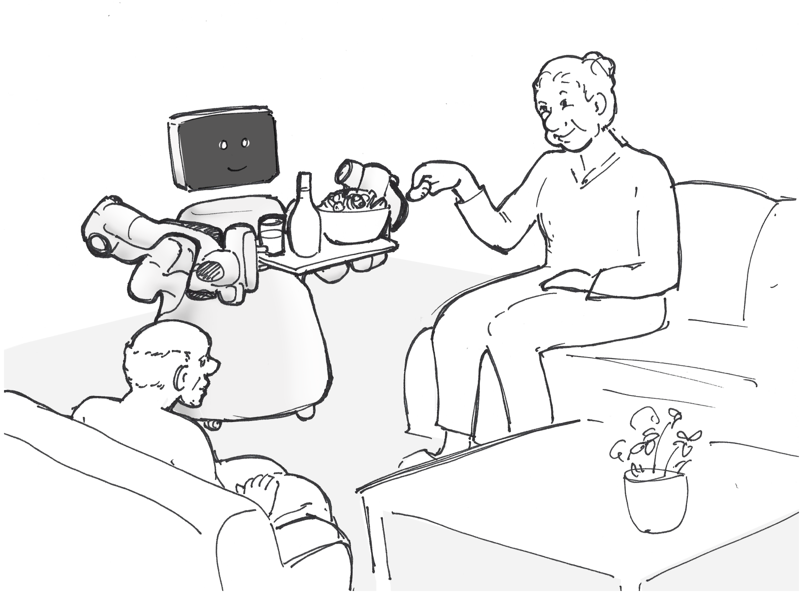


Figure 3.2.: Official KoBo34 illustration for Use Case 2: Kobo serving some snacks and drinks, to encourage people to stay hydrated.

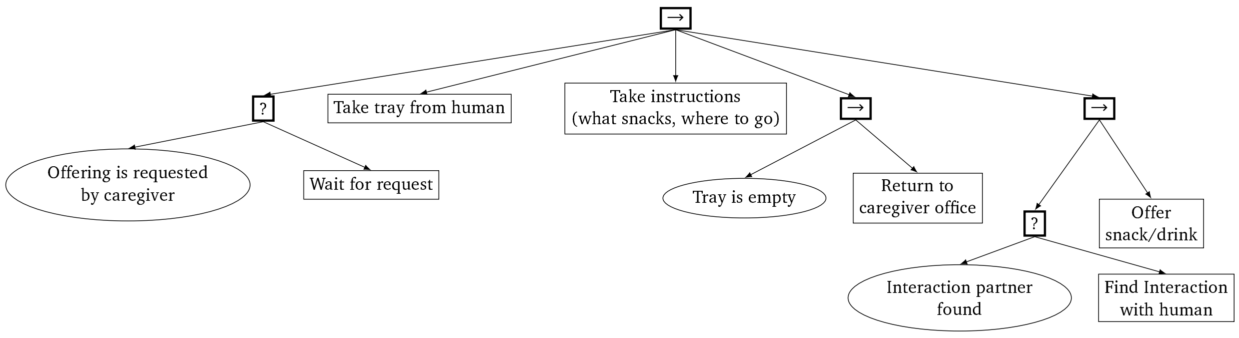

→
?
Take tray from human
Take instructions (what snacks, where to go)
→
→
Offering is requested by caregiver
Wait for request
Tray is empty
Return to caregiver office
?
Offer snack/drink
Interaction partner found
Find Interaction with human

Figure 3.3.: BT proposed by the Kobo project for Use Case 2: Offering Snacks and Drinks. The robot waits for a request by a caregiver. If this request is given, it takes a tray and instructions from a human and searches for an interaction afterwards. Can an interaction partner be found, it offers a snack or drink. If the tray is empty, it returns to the caregiver office. The request can also be terminated at any time, resulting in the execution of the action *Wait for request* again.

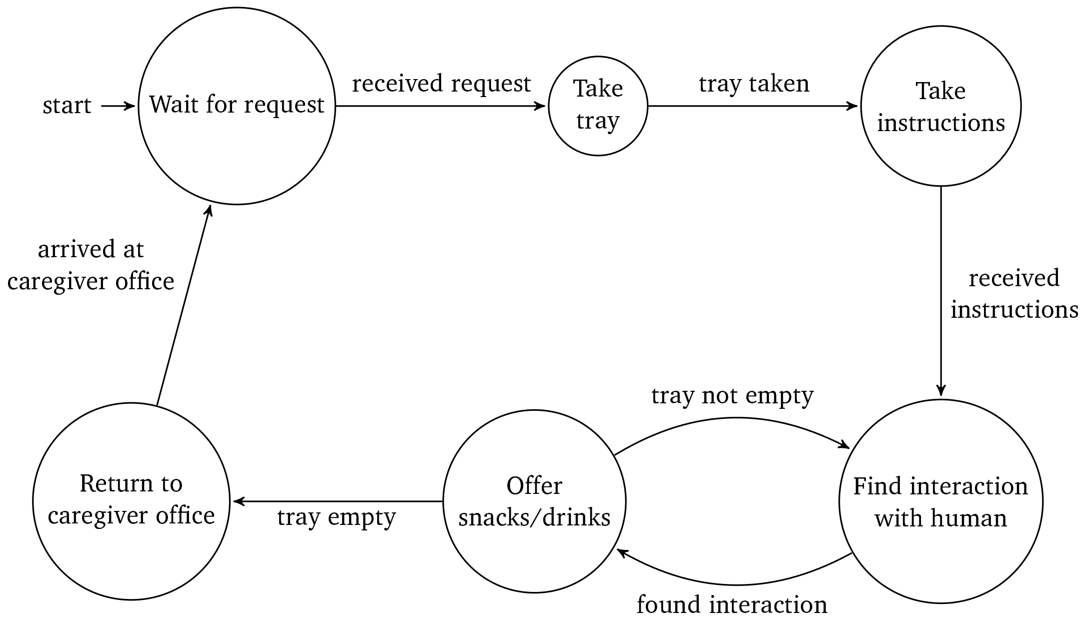


Figure 3.4.: Attempt to recreate the same behavior as shown in the BT in figure 3.3 as a Finite State Machine. The different actions are executed in the individual states. This FSM lacks the reactiveness of the BT, where the request could be terminated at any time.

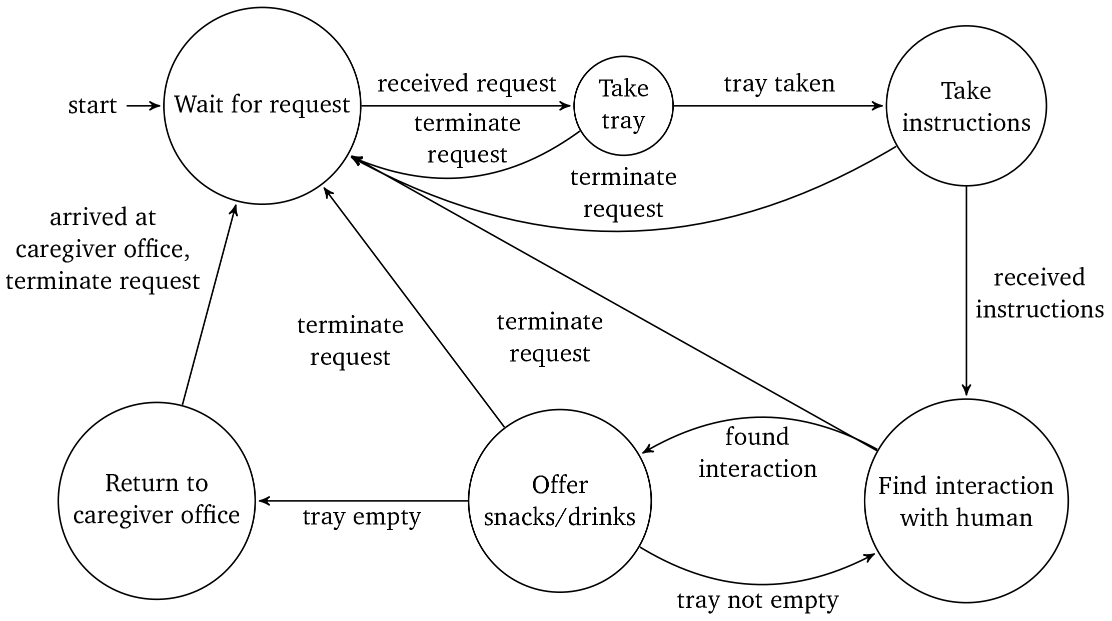


Figure 3.5.: Attempt to add some reactiveness as shown in the BT in figure 3.3 to the Finite State Machine seen in figure 3.4. Several transitions have been added to add the ability to end the request at any given state and to send the robot back to *Wait for request*

3.2.3 Use Case 3: Helping at the Kiosk

The third use case envisions Kobo helping at the kiosk. The seniors could order Kobo to fetch an item or to carry a heavy bag (see also illustration in figure 3.6). This enables seniors with physical disabilities to stay active and keep going to the kiosk, even if they can't carry their purchases themselves. In this environment, it is important that the robot behavior is determinstic, because of the interaction with the elderly person it is helping and the other people also shopping at the kiosk.

Figure 3.6.: Official KoBo34 illustration for Use Case 3: Kobo helping at the kiosk. The robot could fetch items, ordered by elderly people or it could carry a heavy bag for them.

3.2.4 Use Case 4: Kobo Learns

The fourth use case is more abstract than the others and is the key reason for this thesis. It envisions Kobo learning about the people around him and also learning new tasks. Therefore, it has to be able to learn new behavior from demonstrations of elderly persons or caregivers without having to ask a programmer. It should also be able to change an existing behavior while execution. The main reason for this is to enable the addition of new demonstration data. Thereby, the demonstrator could easily observe Kobo's behavior and correct him if necessary. One possibility for Kobo to interact with the people around him is also to ask questions, which is discussed in more detail in section 4.1.2.

3.3 Rationale for Using Behavior Trees

The decision, which behavior structure should be used for the Kobo robot, was mainly based on the prerequisites defined in section 3.1. An overview of all prerequisites and whether they are fulfilled by FSMs and BTs can be found in table 3.1. The demonstration data is modeled as a set of transitions Δ in which the individual states are represented by a finite set of conditions as defined in section 3.1. In a Finite State Machine, the individual states are directly represented by the states of the FSM. Although one FSM state could represent multiple demonstration states. In a Behavior Tree, the state is represented indirectly by the condition nodes present in the BT. Therefore, the state is only evaluated if a condition node is ticked in the BT. Ultimately, both representations can work with the given state model. FSMs and BTs can also both interact with inputs and outputs. The reaction to inputs can be implemented with state changes. FSMs can also be modeled to output signals with every state transition and BTs can create output via *Action* nodes. As shown in section 3.2.2, BTs have the advantage to represent a very reactive behavior with a small number of nodes. In contrast, reactiveness in FSMs has to be modeled explicitly and creates a high amount of redundant state transitions. Therefore, readability of the created behavior is diminished. Another important prerequisite is the possibility to exchange the behavior at run-time. This is easily possible for BTs, if the behavior is changed in between *tick* signals. A BT always

evaluates the whole tree for every *tick* by sending the *tick* signal through the tree until a return value is generated. If the BT structure changes after a returned result of *RUNNING*, the new Behavior Tree is automatically evaluated according to the current state. It is important to note, that the state for a BT, even for BT expansion, has to be modeled externally (see also section 4.1.1). In contrast, the state in a FSM is modeled directly in the behavior structure. If the FSM structure would be replaced while run-time, it has to be ensured, that the last state or an equivalent also exists in the new FSM and is set to be the initial state. Both structures can model deterministic behavior, if it is enforced. They can also both be easily understandable and human-readable if the number of states or nodes is kept low. FSMs are sufficiently expressive for the needs of the Kobo robot and "BTs are at least as expressive as FSMs" [Colledanchise and Ögren, 2017, p. 39]. Both can be used with a stable implementation: *FlexBE*[2] for FSMs and behaviortree.cpp[3] for BTs. As seen in section 2.3, FSMs and BTs can also be generated automatically and for both representations a solution exists, which would work in reasonable time and with reasonable computational effort on the Kobo robot hardware. Finally, both structures provide means of extensibility and modularity. FSMs can be extended by adding new states and transitions and can be modular via *Hierarchical State Machines*. BTs can be extended by replacing nodes with new subtrees, which could also function as modules. However, BTs also have some disadvantages. For example, they need another mindset than for FSMs and the BT tools are less mature than the software available for FSMs [Colledanchise and Ögren, 2017, p. 43].
In conclusion, we decided to continue our work with Behavior Trees. The lack of reactiveness in FSMs, which could only be compensated by more complex state machines, conflicts with the defined prerequisite of an easily understandable behavior structure. While BTs can model a highly reactive behavior, the complexity of the behavior representation can be kept at a minimum. Furthermore, the exchange of the behavior structure while run-time is simpler for BTs, because of the external state model. State matching for a FSM exchange is more complicated and might possibly even fail in some cases.

[2] `http://wiki.ros.org/flexbe`
[3] `https://www.behaviortree.dev/`

Prerequisite	**Finite State Machine**	**Behavior Trees**
Work with given state representation	✓The given state corresponds to states in a FSM (see section 2.1)	✓The state is represented by condition nodes in BT (see section 2.2)
Reaction to inputs/creation of outputs	✓FSMs react to and can create outputs [Harel, 1987]	✓Reaction is possible via action nodes (see section 2.2)
Reaction to unforeseen external changes	✗ No reactiveness (see section 3.2.2)	✓Reactiveness is a key feature of BTs (see section 3.2.2)
Behavioral changes at run-time	✗ States have to be matched to change behaviors at run-time (see section 3.3)	✓BTs can be exchanged in between ticks [Colledanchise and Ögren, 2017, p. 94]
Deterministic behavior	✓Only if DFA logic is enforced (see section 2.1.1)	✓If all actions are deterministic (as defined in section 3.1)
Easily understandable	✓Vague prerequisite, which should be achievable if the number of states and transitions is kept low	✓Vague prerequisite, which should be achievable if number of nodes is kept low [Colledanchise and Ögren, 2017, p. 38]
Expressiveness	✓FSMs can model all our needed behaviors (i.e. FSMs have been used until now)	✓BTs are at least as expressive as FSMs [Colledanchise and Ögren, 2017, p. 39]
Stable implementation	✓FlexBE	✓behaviortree.cpp
Automatic generation	✓Is possible (see section 2.3.1)	✓Is possible (see section 2.3.2)
Computable within reasonable time and computational effort	✓Computation time depends on the kind of automatic expansion and robot hardware	✓Computation time depends on the kind of automatic expansion and robot hardware
Extensibility and modularity	✓This is possible via Hierarchical FSMs (see section 2.1.2)	✓Each subtree can be seen as a module [Colledanchise and Ögren, 2017, pp. 37,38]

Table 3.1.: List of prerequisites for the behavior structure used for the Kobo robot and whether they are fulfilled by FSMs and BTs. For the definition of prerequisites, refer to section 3.1.

4 Automatic Expansion of Behavior Trees

In this chapter, three different approaches for automatic expansion of Behavior Trees are discussed: *Planning and Acting using Behavior Trees* (PA-BT) with given actions and goal conditions, PA-BT with automatic action and goal state detection in demonstration data and our own approach *Automatic action sequence creation and BT specialization* on demonstration data. Furthermore, a pruning algorithm to remove redundant nodes is described.

4.1 Approach 1: PA-BT with Given Actions and Goal Conditions

The first approach is based on *Planning and Acting using Behavior Trees*, already described in chapter 2.3.2. For this technique a set of actions and a goal condition have to be provided. Every action a has a finite set of preconditions p and a finite set of ramifications r. For better comparability, this first attempt is based on the Ball Example described in [Colledanchise and Ögren, 2017, pp. 9-10] (see also figure 2.5). Additionally, for further simplification, the action *Ask For Help* has not been modeled. This example models the behavior of a robot which finds a ball, goes to it, grasps it and places it in a bin (see also Figure 2.5).

4.1.1 PA-BT with Predefined Actions

To provide the PA-BT algorithm with a goal condition and a first set of actions, the preconditions and ramifications of all actions were hard-coded and can be seen in table 4.1. The goal condition was defined to be *Ball Placed*.

Action name	**preconditions**	**ramifications**
Find Ball	-	$BallFound$
Approach Ball	$BallFound$	$BallClose$
Grasp Ball	$BallClose$	$BallGrasped$
Approach Bin	$BallGrasped$	$BinClose$
Place Ball	$BinClose$	$BallPlaced$

Table 4.1.: The hard-coded actions for approach 1 with their preconditions and ramifications.

The inital state is set as well with all conditions returning `FAILURE` and is reset for every new generation. The algorithm initializes the BT with only one condition node of the goal condition (see Figure 4.1). This BT is then executed and if it fails, the last failing condition is replaced by its corresponding atomic BT.

Ball Placed

Figure 4.1.: The initial BT created by the PA-BT algorithm consisting only of the goal condition.

In this case, the condition node *Ball Placed* returned `FAILURE` as specified in the initial state and is therefore expanded by its corresponding atomic BT which consists of all the actions which have *Ball Placed* as their ramification and their individual preconditions (see Figure 4.2). In this case, the condition *Ball Placed* is replaced with a Fallback node with the replaced condition *Ball Placed* and a Sequence node as children. The Sequence node has the action *Place Ball* and its precondition *Bin Close* as children. Therefore, if the ball has not been placed yet, the robot checks if it is next to the bin and if so, tries to place the ball.
This process of expansion is repeated until the BT returns `SUCCESS` or every condition has been expanded already. To achieve this, the algorithm keeps a list of all already expanded nodes.
The capability to also deal with multiple goal conditions (see also Section 6.2) was also implemented. The proposed solution is to change the initial BT to a Sequence node which has all goal conditions as children (see Figure 4.3).

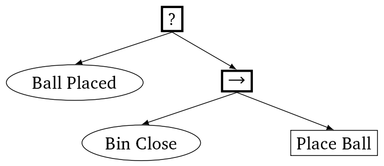


Figure 4.2.: The BT after the condition *Ball Placed* has been replaced by its atomic Behavior Tree. The atomic BT consists of a Fallback node with the replaced condition and a sequence node as children. The sequence node has the preconditions for an action and the action itself as children. For every action, which has the replaced condition as ramification, a Sequence node with its preconditions and the action itself is created.

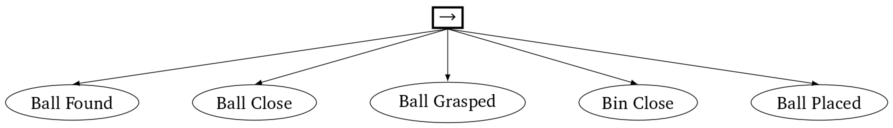


Figure 4.3.: Initial BT for PA-BT with multiple goal conditions. All goal conditions are initialized as child nodes of a sequence node.

4.1.2 Determination of Preconditions and Ramifications of Actions

One major question for this first approach was, how to gain the knowledge about the given actions. One possibility would be to enforce that the programmer of the robot has to define every precondition and ramification of every action. If this knowledge about the world and the effects of its actions were given, a BT could be generated following the approach described in section 4.1.1 by only defining the goal conditions. The presence of all actions and their effects on the environment could be seen as form of simulation. But this would either increase the work of the programmer or limit the robots possibilities.
It is one key demand for this project that the robot should be able to learn while interacting with elderly people and caregivers. Therefore, an intuitive way of teaching the robot has to be found. One way of interaction could be the robot asking for all preconditions and ramifications of his actions. This could only be a solution with a very small state space and could be an interaction not appropriate for seniors. In addition, an unexperienced teacher could possibly forget or define unnecessary goal conditions, which could lead to a different behavior.
To see if these assumptions really hold, a study has been performed. The study consists of three main parts, which can be found in appendix A.1. The first question, intentionally a very open question, is asked to find the first, intuitive approach of the study participant, to teach a robot a simple pick and place task. It is important that this question is not influenced by prior knowledge. Therefore, only a symbolic illustration is given (see figure A.1) for the question: "You want to teach a robot to place a ball in a bin. How would you describe this to it?". Additionally, the participant is asked, if he would have liked further information about the robot to describe this task.
For the second question, the possible actions (Approach Ball, Place Ball, Approach Bin, Find Ball and Grasp Ball) and conditions (Ball Grasped, Bin Close, Ball Placed, Ball Found, Ball Close) representing the state are given in a shuffled order. This question is asked to see, if people would describe the behavior differently if the set of actions and conditions is defined clearly. Additionally, the participant is asked again, if he was missing any information about the actions and conditions.
The third and last part is modeled after the PA-BT approach. For the algorithm to work, the preconditions and ramifications of every action have to be defined to model atomic BTs (see section 4.1.1). Therefore, the study participant is asked to define all preconditions and ramifications of all actions with the five conditions (Approach Ball, Place Ball, Approach Bin, Find Ball and Grasp Ball). This question is asked to see how participants react to this kind of description, to further compare their given definition and to see if a usable behavior would have been generated from it. One further question is asked to find out how many questions of the robot would be tolerable for a user for teaching the robot this task. This question could be helpful to see if an approach would be in an acceptable range of questions.
Lastly, after every main question, the study participant is asked how difficult, on a scale of 1 to 5 (5 being very hard), this particular kind of description has been for her or him. For the results of this study, see section 6.1.

4.2 Approach 2: PA-BT with Automatic Action and Goal State Detection in Demonstration Data

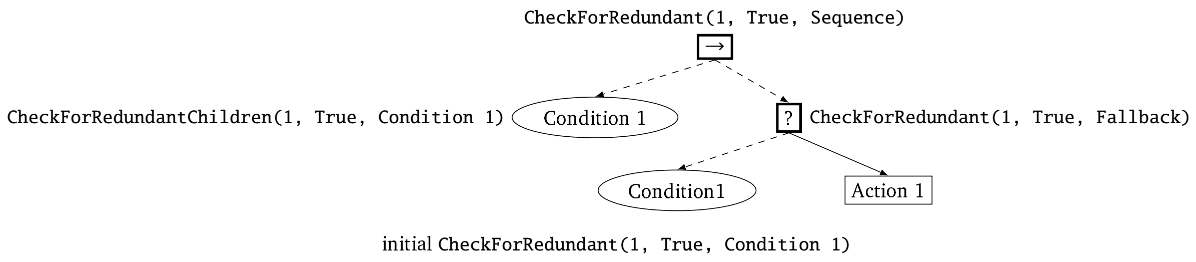
In the following section, our second approach is described. It is also based on PA-BT, but with automatic action detection. Instead of giving the goal state and all preconditions and ramifications of all actions, these are detected from given demonstration data.

4.2.1 Automatic Action Detection

To detect the preconditions and ramifications of an action, a pretty straightforward approach was chosen. For precondition detection, all states s_i, in which the given action a has been applied, are compared. Only if a condition c_j has the same value in all these states, it seems to be a precondition. To learn the ramifications of a given action a, for every (s, a, s') tuple, differences in between s and s' are found. Only if a condition c changes its value in every tuple, it seems to be a ramification of action a. If the demonstration was successful, the last state is marked as a goal state. To find the goal conditions, simply the preconditions of all goal states are determined. This is clearly a very aggressive approach which added several new preconditions. It could certainly be refined in future work.

4.2.2 Pruning

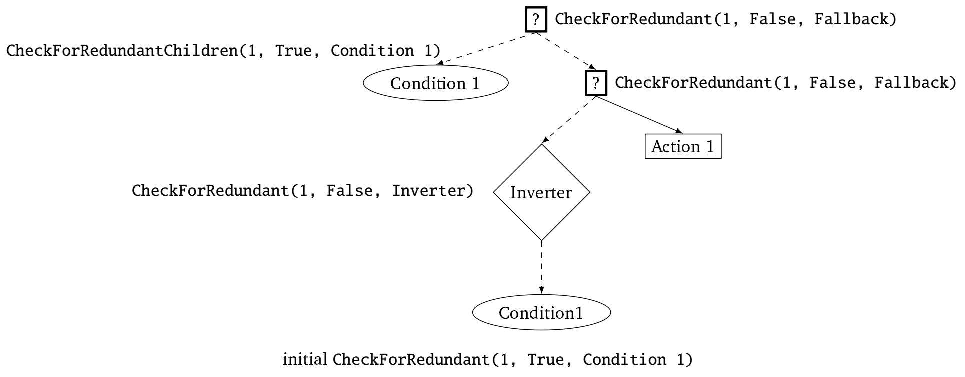
The addition of further preconditions made it necessary to implement a simple pruning algorithm. The algorithm consists of four different algorithms described in Algorithm 4.1 to Algorithm 4.4. The algorithm seen in Algorithm 4.1 starts the pruning for every condition node by recursively checking for every given node, if it has been already evaluated in another node, in case it is reached by the tick signal. As seen, for example in Figure 4.4. Therefore, `CheckForRedundant`$(c_{ID}, c_{value}, node)$ is invoked on the condition node and propagated thereafter to its parent nodes.



CheckForRedundant(1, True, Sequence)
→
CheckForRedundantChildren(1, True, Condition 1)
Condition 1
?
CheckForRedundant(1, True, Fallback)
Condition1
Action 1
initial CheckForRedundant(1, True, Condition 1)

Figure 4.4.: Simple pruning example 1: If Condition 1 connected to the Fallback node is reached by the tick signal, it must have been evaluated as True before, because the same Condition has to have returned SUCCESS already for the Sequence node to continue.

The BT is traversed by `CheckForRedundant`$(c_{ID}, c_{value}, node)$ starting at the given condition node in direction of his parent. The pruning algorithm propagates the ID of the condition and its value to its parent. The parent therefore checks all its subtrees to the left of the subtree, the check for redundancy propagated from, via `CheckForRedundantChildren`$(c_{ID}, c_{value}, node)$. If the parent is a *Fallback* or *Sequence* node, the search for redundancy is propagated directly to its parent. If the parent node is an Inverter, the value of the propagated condition is inverted as well and the pruning is continued with the next parent node. As seen, for example in Figure 4.5.



?
CheckForRedundant(1, False, Fallback)
CheckForRedundantChildren(1, True, Condition 1)
Condition 1
?
CheckForRedundant(1, False, Fallback)
Action 1
CheckForRedundant(1, False, Inverter)
Inverter
Condition1
initial CheckForRedundant(1, True, Condition 1)

Figure 4.5.: Simple pruning example 2: If a *Inverter* node is reached, the pruning algorithm continues with its parent node but with the inverted value of the propagated Condition.

If the parent node is a *Sequence* node, every child to the left of the subtree, the pruning process started from, has to have returned SUCCESS. Therefore, if the propagated condition has been already evaluated in one child node to the left of the *Sequence* node, the given condition node can be pruned. This search is continued recursively until every leaf node has been reached. If one subtree shows that the propagated condition has already been evaluated, the given condition can be pruned, because a *Sequence* node guarantees that all subtrees to the left of the original subtree have been evaluated. If an

Action node is reached, and its ramifications contain the propagated condition, this is also a sign that the given condition can be pruned. If the parent node is a *Fallback* node, that means that all of its children have to have returned `FAILURE` to propagate the tick signal to the given condition. Therefore, if the negation of the given condition has already been evaluated, the given condition can be pruned. The search is continued recursively as well, but in this case, all subtrees have to show that the given Condition can be pruned.
It is possible that this pruning algorithm has to be executed multiple times, because effects in subtrees can only become visible after other nodes have been pruned before. After pruning, nodes with only one child (except *Inverters* and the root node) are removed to further simplify the BT.

```
Algorithm: Pruning of Behavior Trees
Data: an unpruned Behavior Tree $BT$
Result: a Behavior Tree representing the same behavior as the given one, without unnecessary nodes
foreach condition node $c$ in Behavior Tree $BT$ do
    value = $c_{value}$
    if typeOf(parentOf(node)) = Inverter then
        parent = $parentOf(parentOf(node))$
        value = $\neg value$
    else
        parent = $parentOf(node)$)
    if $typeOf(parent)$ = Fallback and CheckForRedundant($c_{ID}, \neg value, parent$) then
        Prune ($c$)
    else if $typeOf(parent) =$ Sequence and CheckForRedundant($c_{ID}, value, parent$) then
        Prune ($c$)
```

Algorithm 4.1: The algorithm used to start pruning. All subtrees, executed before the given node, are checked via `CheckForRedundant` if the given node has been evaluated already in all cases. A distinction is made according to the type of the parent node.

4.3 Approach 3: Automatic Action Sequence Creation and BT Specialization on Demonstration Data

The performed study (see section 4.1.2 and section 6.1) showed that most demonstrators did not focus on clarifying the state but were more focused on showing the sequence of actions used to achieve the given goal. Therefore, a different approach was developed to focus more on action sequences. With this focus, the possibility to model loops was sacrificed, mainly because it never appeared in the given demonstration data.

4.3.1 Automatic Action Sequence Creation

To find the order of actions in demonstration data, all action sequences are extracted from the demonstration data and topologically sorted. Therefore, a directed graph of all actions is generated with directed edges from every action a_i to its successor a_{i+1}. For example, the topological order of the given action sequences (A,C), (B,C) and (A,B) would result in (A,B,C) as seen in figure 4.6. The generation of a topological order is only possible if no cycles exist in the Graph.

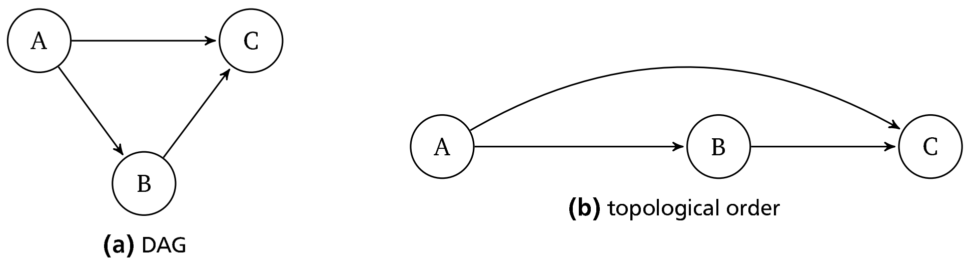


Figure 4.6.: Directed Acyclic Graph (DAG) of three Actions (A,B,C) with edges (A,C), (B,C) and (A,B). The topological order (A,B,C) can be seen in (b). The vertices and edges in (b) are unchanged from (a) and have only been rearranged to illustrate the topological order.

```
Algorithm: CheckForRedundant($c_{ID}$, $c_{value}$, $node$): Checking for Redundancy of a Behavior Tree Node
  (Ascending)
Data: the ID $c_{ID}$ of the condition which is checked for redundancy,
      the value $c_{value}$ of the condition which is checked for redundancy and
      the node $n$ where the redundancy check propagated from
Result: is_redundant: True if the condition with the ID $c_{ID}$ has been found redundant, False otherwise
if typeOf($parentOf(node)$) = Inverter then
    return CheckForRedundant($c_{ID}$, $\neg c_{value}$, $parentOf(node)$)
else if typeOf($parentOf(node)$) = Root then
    return False
else if typeOf($parentOf(node)$) = Fallback then
    check every child node to the left of the given node
    forall child in childrenOf($parentOf(node)$) do
        if child = node then
            break
        else if CheckForRedundantChildren($c_{ID}$, $\neg c_{value}$, $parentOf(node)$) then
            return True
    if no redundancy has been found in the children, check for redundancy one level higher
    return CheckForRedundant($c_{ID}$, $c_{value}$, $parentOf(node)$)
else if typeOf($parentOf(node)$) = Sequence then
    check every child node to the left of the given node
    forall child in childrenOf($parentOf(node)$) do
        if child = node then
            break
        else if CheckForRedundantChildren($c_{ID}$, $c_{value}$, $parentOf(node)$) then
            return True
    if no redundancy has been found in the children, check for redundancy one level higher
    return CheckForRedundant($c_{ID}$, $c_{value}$, $parentOf(node)$)
```

Algorithm 4.2: The algorithm used to check if a redundancy of a given condition can be found via the parent of the given node. For a condition to be found redundant, it has to be evaluated before in all cases. If the parent node is an *Inverter*, the algorithm is executed recursively with the inverted value. The algorithm first checks for redundancy in the children to the left of the given node. If no redundancy is found, CheckForRedundant is executed recursively on the parent node. A case distinction is made for Fallback and Sequence nodes.

The implementation runs the topological sort algorithm on all vertices if they have not yet been visited. For every edge to a vertex which has not been visited, the algorithm is executed recursively and finally the current node is pushed as first element to a stack which is then returned as result. If a topological ordering can be found, there must exist a coherent order in all of the demonstration data.

4.3.2 BT Specialization

The generated topological order of actions is used to create an initial BT, consisting of a *Sequence* node with all actions in the given order as children. Every action a is connected to the *Sequence* node via a *Fallback* node that also has every condition with and without an *Inverter* before a as children, as seen in figure 4.7. This initial BT would return SUCCESS in all cases because every condition is present in its original form as well as its inversion. Therefore, one condition node in every *Fallback* node will always return SUCCESS. To specialize the BT, it is run for the first time on the first state of the first demonstration. If the BT returns SUCCESS it is checked if it also executed the same action sequence. In the first iteration, no action is executed, because for all actions, one condition node to the left of the action, returned SUCCESS. Therefore, the subtree of the first action a_1 that was not executed as expected, is updated. Every condition or *Inverter* that would return SUCCESS in the state where a_1 was expected is removed with the result that in this particular state, the action a_1 would be reached by the *tick* signal and executed.
If, for the example BT seen in figure 4.7, *Action 1* should have been executed in the state where *Conditions 1* and *2* were true and *Conditions 3* and *4* were false, these Conditions (i.e. *Condition 1, 2* and the individual subtrees of *Inverters* connected to *Condition 3* and *4*) are removed, resulting in the BT seen in figure 4.8. After this, the BT is started again with the initial state of the same demonstration. When *Action 1* is now executed, the state is set to the next state in

```
Algorithm: CheckForRedundantChildren($c_{ID}, c_{value}, node$): Checking for Redundancy of a Behavior Tree Node
  (Descending)
Data:  the ID $c_{ID}$ of the condition which is checked for redundancy,
       the value $c_{value}$ of the condition which is checked for redundancy and
       the node $n$ which ist checked for redundancy
Result: is_redundant: True if the condition with the ID $c_{ID}$ has been found redundant, False otherwise

if typeOf(node) = Inverter then
    return CheckForRedundantChildren($c_{ID}, \neg c_{value}, childOf(node)$)
else if typeOf(node) = Condition then
    if $node_{ID} = c_{ID}$ and $node_{value} = c_{value}$ then
        return True
else if typeOf(node) = Action then
    if $node_{ID} = c_{ID}$ and $node_{value} = c_{value}$ then
        forall ramification in $node_{ramifications}$ do
            if $ramification_{ID} = c_{ID}$ and $ramification_{value} = c_{value}$ then
                return True
else if typeOf(parentOf(node)) = Fallback then
    only if all child nodes return True, return True as well
    forall child in childrenOf(node) do
        if ¬ CheckForRedundantChildren($c_{ID}, c_{value}, child$) then
            return False
    return True
else if typeOf(parentOf(node)) = Sequence then
    if one child node returns True, return True as well
    forall child in childrenOf(node) do
        if CheckForRedundantChildren($c_{ID}, c_{value}, child$) then
            return True
    return False
```

Algorithm 4.3: The algorithm used to check if a redundancy of a given condition can be found in a child node of a Sequence or Fallback node. In case the given node is an *Inverter*, the algorithm is propagated recursively with the inverted value. If a condition has the same ID and value as given, the condition with the given ID has been found redundant in this subtree. If an action has the condition with the given ID and value as ramification, the condition with the given ID is found as redundant in this subtree as well. If the given node is a Fallback node, the algorithm is executed recursively on all children. Only if all child nodes find the condition with the given ID as redundant, the algorithm as well finds the condition with the given ID as redundant. If the given node is a Sequence node, only one child has to find the condition with the given ID as redundant.

the demonstration. The procedure is repeated until the BT recreates the action sequence of the demonstration. After this, the algorithm continues with all other demonstrations to specialize the BT to all given demonstration data. If the demonstration data contains an external action, the state is set accordingly. This algorithm does not support pruning in between generations. Redundant conditions have to be kept for further specialization steps.

```
Algorithm: Prune(node): Remove a node which can be pruned
Data: a Behavior Tree $BT$ in which the prunable node should be removed,
      the Behavior Tree $node$ which can be pruned
Result: a Behavior Tree without the pruned node and other unnecessary nodes
if numberOfChildren($parentOf(node)$) = 1 then
    $BT$.remove($parentOf(node)$)
else if numberOfChildren($parentOf(node)$) = 2 then
    if there is only one neighboring node, the parent node can be replaced with it
    $BT$.replace($parentOf(node)$ , $node_{neighbor}$)
else
    $BT$.remove($node$)
```

Algorithm 4.4: The algorithm used to remove a prunable node. If the node is the single child of its parent, its parent is removed as well. If the node has only one neighbor node, its parent is replaced with the neighbor. In all other cases, the parent node has more than 2 children and therefore, the given node can simply be removed.

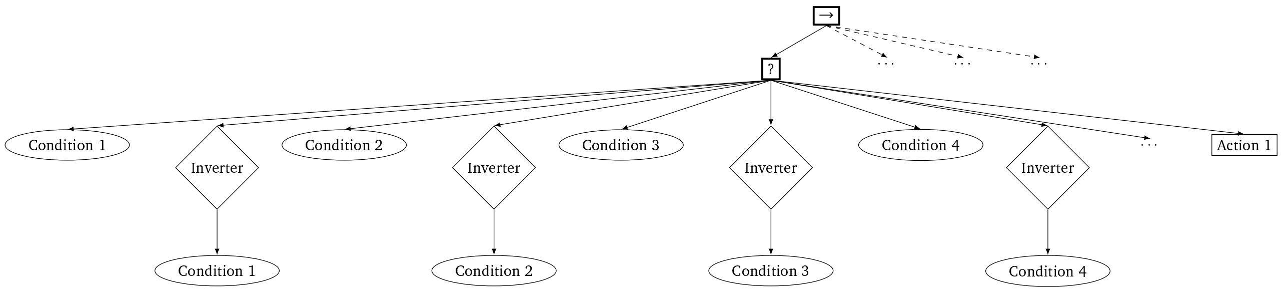


Figure 4.7.: Initial BT for BT specialization on demonstration data. Every action is connected to the root *Sequence* node via a *Fallback* node. This *Fallback* node has all conditions as well as all inverted conditions as children to the left of the action.

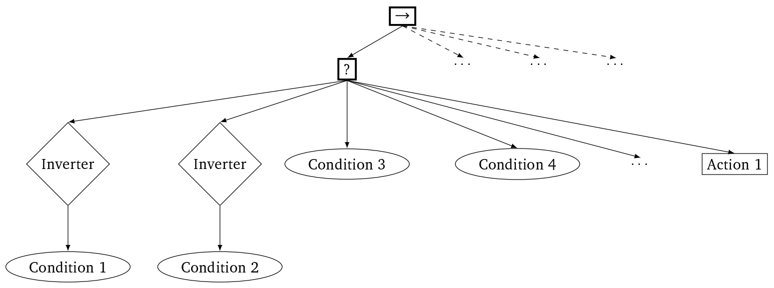


Figure 4.8.: Example BT after the first specialization on demonstration data. All nodes which would return SUCCESS before the expected action are removed. In this case, *Action 1* has to be executed if *Condition 1* and *Condition 2* are true. Therefore, these condition nodes are removed. Additionally, *Condition 3* and *Condition 4* have to be false to execute *Action 1*. To achieve this, the subtrees with the inverted conditions are removed. The resulting BT executes *Action 1* in the desired state.

5 Implementation and Setup

To work with Behavior Trees, the stable implementation `behaviortree.cpp`[1] has been chosen. The IAS department and therefore the Kobo robot is implemented using the *Robot Operating System*[2] (ROS) [Quigley et al., 2009]. An overview of the implemented system architecture can be seen in figure 5.1. The `BT Generator` and its Graphical User Interface (GUI) is implemented in *Python* and starts a `SimpleActionClient`, provided by the `actionlib` package, which waits for the `BT Server` (`behavior_tree_cpp`) to start. Once this server is started, clients can send goals, containing the path to a Behavior Tree XML-File, which is then executed via `behaviortree.cpp`. The `BT Server` returns the return value of the executed BT as an integer value: 0 for `SUCCESS`, 1 for `FAILURE` and 2 for `RUNNING`. `behaviortree.cpp` executes the tree nodes according to the BT logic. The nodes are all modeled as `CustomActionClientTreenodes`, which possess a node type (i.e. *Action* or *Condition*), an ID and a unique ID (UID). The ID, defines which condition or action should be evaluated or executed. The UID is for exact identification of a node, for example to replace an exact node or for debugging purposes. To model the current state, the `BT Generator` initializes a `State Server` which is also implemented as `SimpleActionServer`. Therefore, the `CustomActionClientTreenodes` send the node type and the ID of the node as a goal to the `State Server`, which returns the value of this node for the current state. In particular, it returns an integer value representing `SUCCESS` (0) for a true condition, `FAILURE` (1) for a false condition and `RUNNING` (2) for any action. In our implementation, actions can only return `RUNNING`, but change the state according to their ramifications. Therefore, the BTs have to be modeled to only execute an individual action, if the inverted ramifications are preconditions. Otherwise, the BT could get stuck in a loop, where the same action is executed every time. This prerequisite of using explicit success conditions is a design decision, based on the design principles described by Colledanchise [Colledanchise and Ögren, 2017, pp. 45-46], to improve readability.

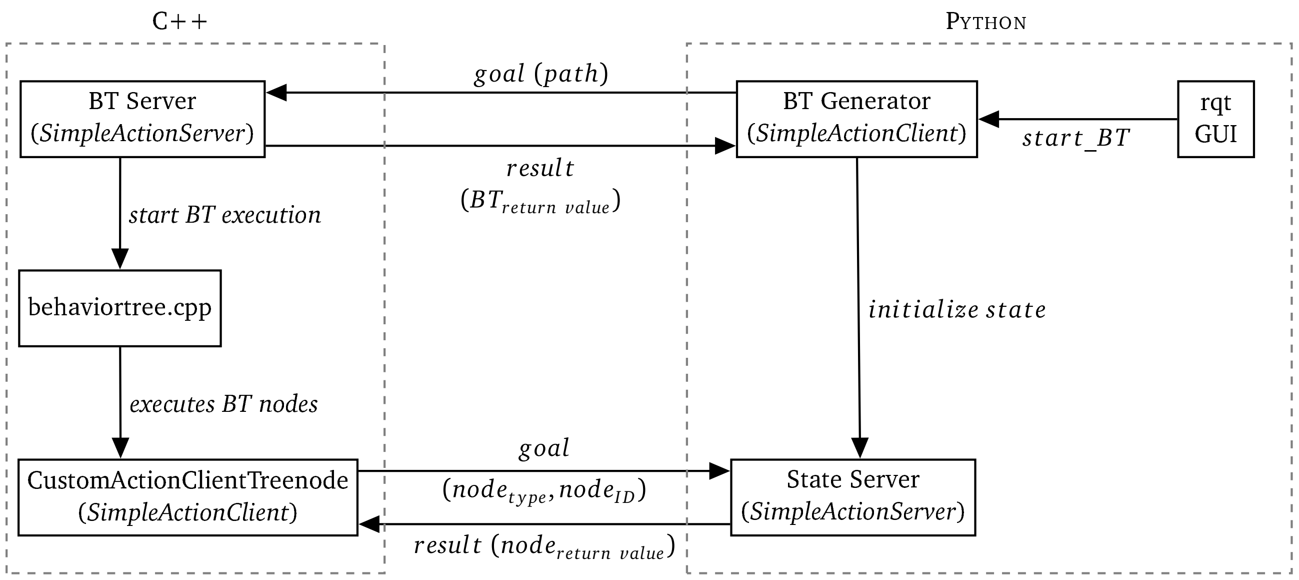


Figure 5.1.: Overview over the system architecture built on the ROS framework. The implementation is separated into a C++ and a Python part. The two parts communicate via two Action Servers and individual Action Clients. The GUI, which starts the `BT Generator`, is implemented in Python. The `BT Generator` sends the path to a Behavior Tree to the `BT Server` which then executes the BT via the `behaviortree.cpp` framework. Additionally, the `BT Generator` can set the current state of the `State Server`, which is also implemented in Python. The BT framework executes the BT nodes according to the BT logic. Every BT node is an Action Client which sends its ID and node type as a goal to the `State Server`. The `State Server` returns the value of the Condition or Action according to the current state. When the BT execution is finished, the `BT Server` returns its return value to the `BT Generator`.

[1] `https://www.behaviortree.dev/`
[2] `https://www.ros.org/`

The demonstration data is saved as comma-separated values (CSV). Each line represents one state, with the conditions ordered by their ID. The truth value of a condition is given as an integer (0 for *False* and 1 for *True*) and the individual values are separated by semicolons. The last value in one line is always the action ID of the action which has been observed in this state. The action ID *0* implies that a goal state has been reached. The action ID *-1* stands for an external action. The BTs are saved in Extensible Markup Language (XML) following the design used by `behaviortree.cpp`.
The `BT Server` is started by the following command: `roslaunch bt_server bt_server.launch`. The `BT Generator` is started via `rqt`.
The *rqt* GUI (seen in figure 5.2) has been implemented for better usability and enables the user to load different BTs and to test them. The initial state can be set via the *State* checkboxes. If a demonstration is chosen from the dropdown list, the preconditions and ramifications for the found actions and the goal conditions are automatically detected. The goal conditions can also be set by the *Goal Conditions* checkboxes. These checkboxes have three states: *Unchecked* means that this condition is no goal condition, *Partially Checked* corresponds to the case, where this condition has to be false in the goal state, whereas *Checked* stands for this condition to be true in the goal state. By pushing the button *Approach 1: Start Expansion via PA-BT with given actions and goal conditions*, the expansion using the method *Planning and Acting using Behavior Trees* is started with the hard-coded action-set seen in table 4.1 and the goal conditions set by the goal condition checkboxes. Clicking on the button *Approach 2: Start Expansion via PA-BT with automatic action and goal condition detection* starts the expansion with the chosen demonstration data, described in section 4.2. The individual expansion generations are saved in the *trees* folder of the `BT Server`. Via *Expand via PA-BT with additional demonstration data* a folder with additional demonstration data can be defined and a new expansion with the enlarged data set is started directly. To expand a BT as described in approach 3 (see section 4.3), the button *Approach 3: Create BT via Automatic action sequence creation and BT specialization* has to be pressed. For this approach as well, additional demonstration data can be added via *BT specialization with additional demonstration data*. In this case, the specialization algorithm continues to specialize the current BT only with the given additional demonstration data. The path to the current tree is shown below the goal conditions and this BT can be pruned by clicking on the button *Prune current BT* (see also section 4.2.2). The pruned BT is saved as new generation in the *trees* folder. The last button, labeled *Test BT over demonstration data*, is used to test a BT over the demonstration data, chosen from the *Demonstration* dropdown list. For all demonstrations, the expected action sequence and the executed actions are compared. If the right action is executed, the state is set according to the demonstration data. If another action as expected is executed, the test is terminated unsuccessfully. If an external action is part of the demonstration data, the external action is executed and the state is set accordingly. The results are saved in the same location as the given BT as a text file.

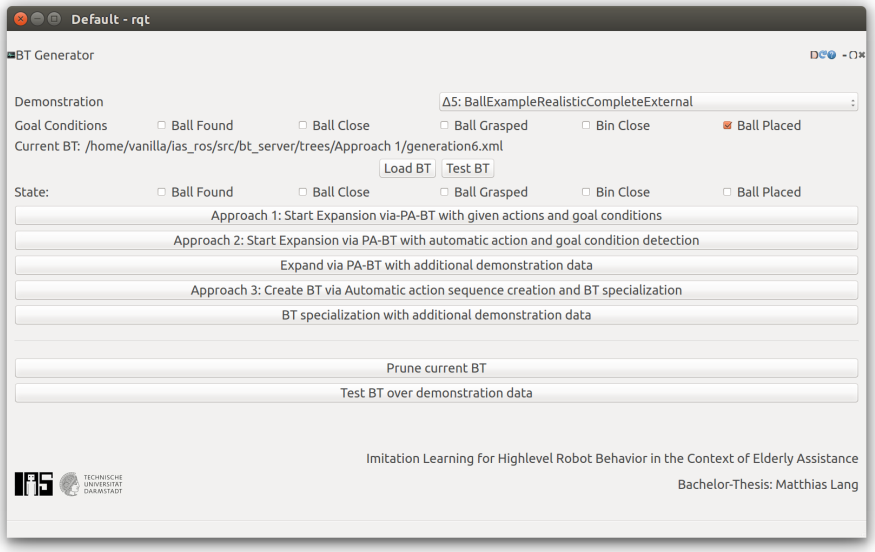

Default - rqt
BT Generator
Demonstration: Δ5: BallExampleRealisticCompleteExternal
Goal Conditions: ☐ Ball Found ☐ Ball Close ☐ Ball Grasped ☐ Bin Close ☑ Ball Placed
Current BT: /home/vanilla/ias_ros/src/bt_server/trees/Approach 1/generation6.xml
Load BT | Test BT
State: ☐ Ball Found ☐ Ball Close ☐ Ball Grasped ☐ Bin Close ☐ Ball Placed
Approach 1: Start Expansion via-PA-BT with given actions and goal conditions
Approach 2: Start Expansion via PA-BT with automatic action and goal condition detection
Expand via PA-BT with additional demonstration data
Approach 3: Create BT via Automatic action sequence creation and BT specialization
BT specialization with additional demonstration data
Prune current BT
Test BT over demonstration data
Imitation Learning for Highlevel Robot Behavior in the Context of Elderly Assistance
Bachelor-Thesis: Matthias Lang
TECHNISCHE UNIVERSITÄT DARMSTADT

Figure 5.2.: The final Graphical User Interface (GUI) of the implementation in rqt. BTs can be loaded and tested directly. Demonstration data can be chosen from a dropdown menu and goal conditions are detected automatically but can also be changed via checkboxes. BTs can be loaded, tested and expanded via the three approaches proposed in this thesis: PA-BT with given actions and goal conditions, PA-BT with automatic action and goal condition detection from demonstration data and automatic action sequence detection and BT specialization. For the approaches 2 and 3, new demonstration data can be added afterwards. The current BT can be pruned and tested over the given demonstration data.

6 Experimental Evaluation

This chapter first presents the results of a study conducted to compare the way people would intuitively teach a robot and further discusses different results with all three proposed approaches. The generated BTs are compared by complexity as well as the amount of redundant nodes and are tested against demonstration data.

6.1 Study about Ways of Robot Teaching

To find an intuitive and appropriate way for elderly people to teach the robot, a study has been performed. There were 8 participants in a range from 21 to 89 years old. There are several possible ways of interaction with a robot to teach it new behavior. For example, the interaction could be verbal in natural language or by direct or indirect demonstration. To avoid any influence on participant's answers, a very open question has been chosen to begin the study.
The intuitive approach of all participants was to focus on the actions the robot has to perform to achieve its goal. For example the answer of participant 1: “Extend your right hand down. Open your hand. Close your hand again, so that you grasp the ball. Lift it and turn in the direction of the bin. Open your hand again and drop the ball”. Only participants 1, 4 and 7 intuitively included conditions for the execution of some actions: “Close your hand again, so that you grasp the ball” or “And if he finds the ball, he should grasp the ball with his hand” and “First of all, the robot has to know where the ball is”. Participant 1 even demonstrated the movements physically. A robot capable of learning from demonstration could have used this information as well. Participant 6 even tried to change the position of the bin, to simplify the task for the robot. 5 participants did not need further information about the robot for this kind of description. The information requested by other participants concerned the movement possibilities of the robot (Participant 2: “I don't know how he can move.”), its functions (Participant 3) and for example how far the robot can look and how it knows, what a ball is (Participant 4). The participants who did not need more information also referred to the given illustration. For example, participant 5: “The picture was sufficient for me”. On average, participants took 39 seconds for this kind of description and did not find it very difficult (≈1,4 on a scale from 1 to 5).
For the second question, a clear set of actions and conditions was given. But even after knowing the exact capabilities of the robot, some participants did not change their description from the answer given to the first question. For example, participant 1 took several seconds to consider possible changes, but finally did not change the given definition. Even after clarifying that the robot has no action to stretch out its arm, the participant did not change the description. However, when asked, if he thought that the behavior would work like that, the participant responded positively. Participant 3, 5, 6, 7 and 8 tried to find the logical order of the given actions. Participant 7, for example: “Always like that: Check if it is already done, if not, do it. And then work through step by step. But somehow backwards”. Only participant 2 and 8 directly included the conditions: “Then, check if the ball is close to you” and “Next, approach the ball, grasp ball, then check if the ball is grasped”. After asking if the given conditions were relevant as well, almost all participants gave a more detailed description including the conditions, except participant 6, who was convinced that the robot would reproduce the given behavior without giving instructions concerning the conditions. Interestingly, the clarification of the action and condition set, spawned more new questions than the more open first question. Participant 1, for example, did not see the need for further information after the first question, but asked several questions about the robot's movement capabilities and its senses, after the second question. Furthermore, the term “Close” had to be clarified multiple times, which was mixed up with “Findable” by some participants. This shows the importance of clear language. In case, the participants would have to give these information directly to a robot, clear definitions of all actions and conditions have to be given. Additionally, the question was posed, if the robot's actions always succeeded. This would have to be a part of the action definition as well. The introduction of the conditions representing the state also led to the reflection of other start states: “I am thinking that the robot could have grasped the ball already. And if you think it through, he could have finished already as well”.
This kind of description was found more difficult (≈2,7 on a scale from 1 to 5) and it took on average over three times as long as the first one (2 minutes and 18 seconds). Participants got more accustomed to this kind of definition. For example participant 5: “I am beginning to understand this” and “This was kind of extensive now. To always question it once more”. Some participants even gave a difficulty level of 4: “Rather 4, because it is very confusing”.
The third question was answered by filling in a table of preconditions and ramifications for every action. Therefore, the given answers will be compared with the action definition used to implement approach 1, seen in table 6.1. One major issue for this kind of description was to explain the concept of preconditions and ramifications to almost all participants, like for example participant 5: “Because I had to understand how this works, but then actually, it is quite easy”. Participants 1, 2, 3, 6 and 8 needed even further clarification and some found it hard to define actions like this

(Participant 6: “Very strange to think like this”). It is worth mentioning that the ages of these participants are in the range of 56 to 89, whereas the three participants who did not need further explanation, are aged between 21 and 30. For some participants one major problem was the temporal order of preconditions and ramifications. It had to clarified multiple times that preconditions were checked before and that the ramifications were checked after the execution of an action. Some also did not restrict the preconditions and ramifications to the given set of conditions. Even after clarifying this, participant 1 gave “He has to move his arm” as precondition for the action *Find Ball*. Some participants found it hard to concentrate only on conditions and continued to give actions as answers. For example, participant 2 when asked directly for the observations of the robot: “Then he has to stoop”. In contrast, participant 3 even gave new conditions when asked for the preconditions of the action *Approach Ball*: “That the path for him to go there is free”, and questioned the limits of the robot’s abilities: “For example, if the ball is lying on a shelf, where he cannot grasp it”. On the other hand, some conditions were found as implicitly given (Participant 7: “Because, if the ball is in the hand, it cannot be in the bin. That’s what I, as a human, would say.”) or obvious (Participant 2: “I thought that was obvious.”). Some conditions also were not mentioned, because they were checked in an action before already, like participant 3: “We checked this above already”. Participants 1, 2, 3 and 8 mixed the actions and either described the next action already or did not see every action as separated from the others. This had to be explained multiple times, for example for participant 3: “Oh, so that means that before I go to the bin, I have to check again, did he execute the last function, did he grasp the ball correctly”. The definition of “Close” was also discussed multiple time while answering this question, further showing the need for the use of very precise language.

Action	**Preconditions**	**Ramifications**
Find Ball	¬BallFound	BallFound
Approach Ball	BallFound, ¬BallClose	BallClose
Grasp Ball	BallFound, BallClose, ¬BallGrasped	BallGrasped
Approach Bin	¬BinClose	BinClose
Place Ball	BallGrasped, BinClose, ¬BallPlaced	BallPlaced

Table 6.1.: Table of preconditions and ramifications for given actions, generating a usable BT. The study participant’s answers were compared with this definition.

The definition of preconditions and ramifications, seen in table 6.1, has been used to generate a behavior with the PA-BT approach. There are multiple differences to the given answers. If a participant did not give any ramification for an action, the concept of ramifications always was explained once more. But even with this explanation, some participants did not specify ramifications for all actions. For example, participant 3 gave no ramifications or preconditions for the action *Approach Ball*. It was difficult for some participants to understand the temporal order of preconditions and ramifications. Participants 1 and 2, for example, even gave the condition *Ball Found* as precondition for the action *Find Ball* and the condition *Bin Close* for the action *Approach Bin*. Almost all participants used negations of the given conditions except participants 1 and 8. The concept of defining the negation of the ramification as a precondition was used by some (for example ¬*Ball Placed* as precondition for the action *Place Ball* by participants 2 and 5) but was only used rarely, except from participants 5 and 7. In contrast, some participants even defined the same condition as precondition and ramification (Participant 1: *BallFound*, *BallClose*, *BinClose*, *BallPlaced*; Participant 2: *BinClose*, *BallGrasped*; Participant 3: *BallFound*). As described before, the separation of actions was not respected by all participants. Participant 7 for example defined ¬*BallGrasped* and ¬*BallPlaced* as precondition for the action *Find Ball*. This stems from connecting all actions to one unified task. If the robot only uses this action for this individual task, it is not necessary to try to find the ball, if the ball is already placed in the bin. However, the condition that the robot can only try to find the ball if it is not placed in the bin, may be too restrictive. Participants 3, 4 and 7, for example, defined *BallGrasped* as precondition for the action *Approach Bin*. The action to approach the bin is independent of the status of the ball. However, if seen in the context of the whole behavior, the robot should never approach the bin without the ball in its hand. In case, the robot executes this action in another context, but still with the precondition *Ball Grasped* for the action *Approach Bin*, it would be necessary to grasp a ball to be able to approach the bin. Participant 2 even defined all conditions as ramifications for the action *Place Ball*. This resembles the proposed behavior by Colledanchise et al. [Colledanchise and Ögren, 2017], already discussed in section 4.1.1. In contrast, participants 3 and 7 defined ¬*BallGrasped* as ramification for *Place Ball*. This was not modeled in the proposed action set, seen in table 6.1, but could be a reasonable addition. If the robot places the ball in the bin, it should not be grasping it after the action. When applied to the PA-BT approach, only 4 out of 8 action definitions could generate a BT. The proposed definitions by participants 1 and 2 could not be used because of similar conditions as precondition and ramification for one action. This conflict could not be resolved. Furthermore, the

action sets defined by participants 3 and 6 contain actions without ramifications. Therefore, crucial atomic BTs could not be generated. The definitions by participants 4, 5, 7 and 8 show only minor differences to the proposed action definition and could be used to generate usable BTs, which can be found in the figures A.2 to A.5. It is notable that the four not usable definitions are by the four oldest participants. In general, this kind of description was found as more difficult (≈3,4 on a scale from 1 to 5) and took over 5 times longer than the second kind of description (13 minutes 32 seconds on average). Participants described it as cumbersome (Participant 2) and had to get used to it (Participant 8: "5, because I had something different in my had. Another logic." or participant 3: "This was quite difficult. I think I misunderstood everything a little bit."). The participants would accept between 2 and 10 questions by a robot to teach it this task. On average, they would accept 5 questions.
In summary, it can be said that all participant's descriptions concentrated more on actions than on conditions. Even the first, very open question, was answered with action sequences. This kind of description was found very easy and took only 39 seconds on average. For the second question as well, most participants concentrated on the actions. Only two participants directly included the conditions in their description. The more precisely defined action set spawned even more questions about the robot. Furthermore, multiple questions, for example about the definition of "Close", showed the necessity for a clear language and very precise definition of the robot's capabilities. The third question, even more adapted to the PA-BT approach, was found more difficult and took more time. Many participants had to get accustomed to the concept of preconditions and ramifications. Only 4 out of 8 definitions could be used to generate a usable BT. The fact that these definitions were given by the four youngest participants and that 3 of those did not need further explanation of the concept, suggests that age could be a factor, but cannot be verified with a small study like this. Furthermore, the participants would only accept 5 questions by the robot on average for this task.

6.2 Planning and Acting (PA-BT) Approach with Given Actions and Goal Conditions

The proposed Ball Example showed one major difference between hand-modeled BTs and automatically generated ones: the importance of correct goal conditions. A first approach with given actions and the goal condition *BallPlaced* did not result in the same BT as proposed by Colledanchise and Ögren [Colledanchise and Ögren, 2017]. In the proposed BT the robot checks first if it has found the ball and executes the action *Find Ball* accordingly. Afterwards, the robot continues to check for the Condition *Ball Close*, *Ball Grasped*, *Bin Close* and the last Condition checked is *Ball Placed*. In contrast, the automatically generated BT checks for these conditions in reversed order (see figure 6.1). This difference originates from a different set of goal conditions. With the generated behavior, the robot checks first if the ball has already been placed. If this is the case, the behavior stops, because the goal condition has already been reached. The generated behavior is in fact a logical result for this problem. One Sequence node, connected with the action *Find Ball* could be removed, but this would not change the resulting behavior. In contrast to this generated behavior, the proposed BT of Colledanchise and Ögren [Colledanchise and Ögren, 2017] would not be finished if the Ball is already placed. Instead, the robot would first check if it can find the ball, then go to the ball, grasp it, go to the bin and place it again. All condition nodes have to return SUCCESS for the BT to succeed. This means that all five conditions have to be goal conditions. On the other hand, the automatically generated BT lacks the reactiveness of the proposed one. If the condition *Bin Close* returns SUCCESS, the robot will not even check if it has grasped the ball. It would try to execute the action *Place Ball* but the action would not have the expected effect on the condition *Ball Placed*. In this case, the robot would be stuck in a loop, trying to place a ball it never grasped. This result shows that both BTs don't model a realistic behavior. We approached this problem by working with demonstration data, which will be discussed in further detail in section 6.3.
Another important insight gained by this first generated BT is, that the algorithm should be able to work with multiple goal conditions. The implementation of this is discussed in further detail in section 4.1. The PA-BT generation starting with a Sequence node with all goal conditions (see figure 6.2) is then executed and expanded as before as seen in figures 6.3 to 6.7.
This approach generates several redundant condition nodes, which can be pruned with the pruning algorithm proposed in section 4.2.2. Generation 5 after pruning (figure 6.8) of this approach is exactly the same BT as proposed by [Colledanchise and Ögren, 2017].

6.3 PA-BT with Automatic Action and Goal State Detection in Demonstration Data

There were 5 different demonstration sets created to test the algorithm. The demonstration set Δ_1 consists of 32 individual demonstrations, where we tried to recreate exactly the behavior of the proposed BT of [Colledanchise and Ögren, 2017, p. 10]. The key idea behind this set was to try to recreate the proposed BT, where only one goal state with *Ball Found*, *Ball Close*, *Ball Grasped*, *Bin Close* and *Ball Placed* is accepted. The set is state space complete, i.e. all 32 possible states are present. Demonstrations Δ_2 are also modeling every possible state, but the approach with this set changed to model a more realistic behavior. That means that the robot does not try to grasp the ball, if it is already placed. Therefore,

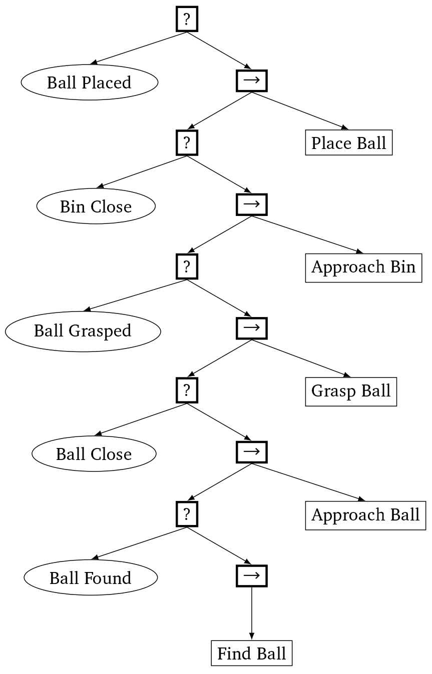


Figure 6.1.: Approach 1: BT generated with the PA-BT approach with given actions (seen in table 4.1) and only *BallPlaced* as a single goal condition.

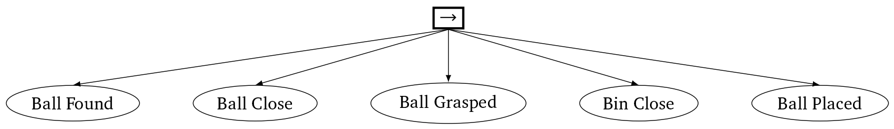


Figure 6.2.: Initial BT with all given goal conditions. For this behavior to succeed, all Conditions have to succeed. The Condition nodes are all children of a Sequence node and are evaluated (i.e. receive the tick signal) from left to right.

not only one, but 16 goal states exist. To further diversify the demonstration set, two special cases are added as well. In case, the robot already stands near the bin (i.e. *Bin Close* is returning `SUCCESS`), but has to go to the ball first, there are two possible outcomes. The condition *Bin Close* could stay unchanged, or it could change to `FAILURE` because the robot moved too far away from the bin. Finally, Δ_2 consists of 34 individual demonstrations. Δ_3 and Δ_4 are subsets of Δ_2. Δ_3 consists of 8 demonstrations and Δ_4 is even smaller with only two demonstrations. These demonstration sets have been created to test the ability to generate a Behavior Tree with only very little information available. Δ_5 is an extension of Δ_2, with added external actions. There are 9 additional demonstrations in which the robot is dropping the ball before and after going to the bin, the ball is removed from its sight, the bin is moved after approaching it and the robot looses the ball location. Furthermore, initial external actions, before the first action are included. Therefore, Δ_5 consists of 43 individual demonstrations. This demonstration set is used to show the capabilities for reactive behavior and for the expansion of reactive behavior.

The simple condition and precondition detection led to the addition of several implicit preconditions, which were not present in the first model. For example for demonstration Δ_1, the precondition detection algorithm added the precondition $\neg$*FoundBall* for the Action *FindBall*. This is logical and necessary, but was not present in the first model of preconditions and ramifications. Therefore, the generated atomic BTs have been added more condition nodes. The expansion, seen in figures 6.9 to 6.14, creates 41 nodes, whereas our first approach with a given action set only created 24 nodes (see figure 6.7). The amount of redundancy is most obvious after pruning this BT (seen in figure 6.15), which results in exactly the same structure as in figure 6.8, which only consists of 16 nodes. This underlines the need for a good pruning algorithm.

The BT created from Δ_2 can be seen in figure 6.16. It models the desired behavior with 44 nodes, but only 17 nodes are retained after pruning, which makes it quite easily readable. The condition *Ball Placed* is checked first, because it is the only goal condition. If this condition is not met, a sequence is started, where it checks first if it has grasped the ball already. If the ball is not grasped, another sequence is executed, which is quite similar to the first part of the proposed BT by [Colledanchise and Ögren, 2017]. The automatic action detection resulted in the same action set for Δ_5 as well and therefore created the exact same BT, seen in figure 6.16.

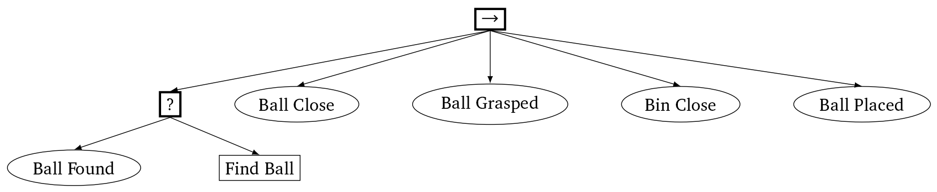


Figure 6.3.: Approach 1: PA-BT with multiple goal conditions: Generation 1. The initial BT, seen in figure 6.2 has been expanded. The Condition node *Ball Found* was replaced with its corresponding atomic BT

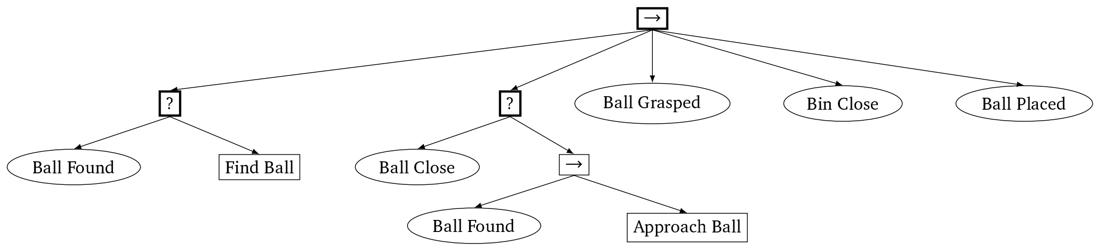


Figure 6.4.: Approach 1: PA-BT with multiple goal conditions: Generation 2. The BT of the previous generation, seen in figure 6.3 has been expanded. The Condition node *Ball Close* was replaced with its corresponding atomic BT

The BTs created from Δ_3 and Δ_4 are more complex. The generation based on Δ_3, seen in figure 6.17, consists of 32 nodes after pruning (53 before pruning) and the BT based on Δ_4, seen in figure 6.18, even consists of 36 nodes (57 before pruning). It is to note that although Δ_3 consists of 4 times more demonstrations than Δ_4, the difference in the amount of nodes is only 12.5%. The pruning algorithm removed 37% to 61% of tree nodes. On average, 50% of created nodes were found to be redundant.
Another evaluation measure was the proportion of the demonstration data, which could be recreated with the given demonstration data. Therefore, the created Behavior Tree was run and compared with given demonstration data. If the right action is executed, the state changes to the next demonstration state. Only if the BT executed the exact same action sequence as in the demonstration, the sanity check is counted as successful. The results of this evaluation can be seen in table 6.2.

Tested on / *Generated from*	Δ_1	Δ_2	Δ_3	Δ_4	Δ_5
Δ_1	32/32 (100%)	13/34 (≈38%)	8/8 (100%)	2/2 (100%)	21/43 (≈49%)
Δ_2	26/32 (≈81%)	34/34 (100%)	8/8 (100%)	2/2 (100%)	43/43 (100%)
Δ_3	11/32 (≈34%)	13/34 (≈38%)	8/8 (100%)	2/2 (100%)	21/43 (≈49%)
Δ_4	6/32 (≈19%)	8/34 (≈24%)	5/8 (≈63%)	2/2 (100%)	13/43 (≈30%)
Δ_5	26/32 (≈81%)	34/34 (100%)	8/8 (100%)	2/2 (100%)	43/43 (100%)

Table 6.2.: The results of the sanity checks on different demonstrations for the BTs generated via PA-BT with automatic action detection. Every result represents how many demonstrations were tested successfully divided by the amount of demonstrations the BT was tested on.

The first observation of table 6.2 is that the generated BTs can always completely recreate the demonstration data they were generated with. This is very important to create consistent behavior. Another important remark would be that even

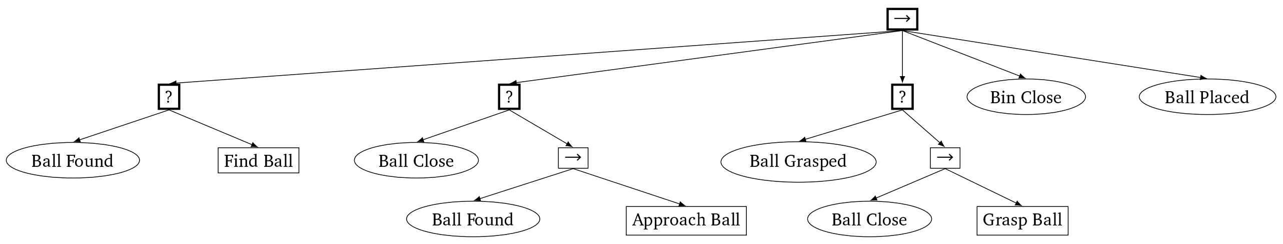


Figure 6.5.: Approach 1: PA-BT with multiple goal conditions: Generation 3. The BT of the previous generation, seen in figure 6.4 has been expanded. The Condition node *Ball Grasped* was replaced with its corresponding atomic BT

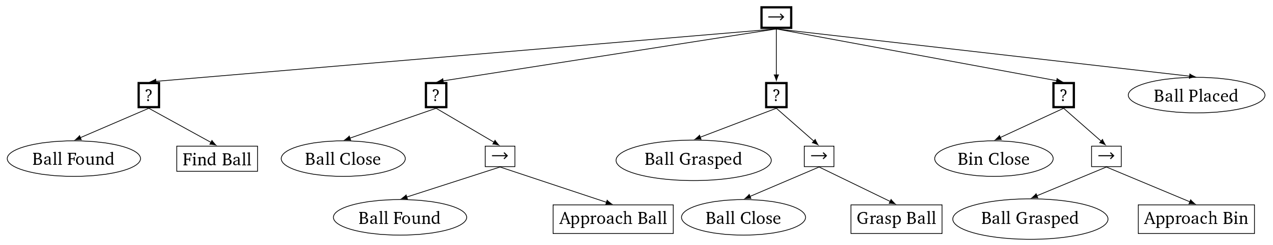


Figure 6.6.: Approach 1: PA-BT with multiple goal conditions: Generation 4. The BT of the previous generation, seen in figure 6.5 has been expanded. The Condition node *Bin Close* was replaced with its corresponding atomic BT

though Δ_1 was created with a different behavior in mind, the generated BT performs very well on Δ_3 and Δ_4. Further inspection showed that the automatically detected goal conditions for Δ_3 and Δ_4 are the same as for Δ_1. This means that the lack of demonstration data could add unintended goal conditions. In contrast, the BTs created from Δ_3 and Δ_4 perform only very poorly on Δ_1. The BTs based on Δ_2 and Δ_5 are the same and therefore perform equally. Both BTs can recreate successfully every demonstration with external actions. The BTs generated from Δ_1, Δ_3 and Δ_4 perform even better on Δ_5 than on Δ_2. This could be explained by the nature of the demonstrations which were added to Δ_2 to create Δ_5. Most of these demonstrations favor an execution in which all conditions are true at the goal state, which is the single goal state for Δ_1, Δ_3 and Δ_4. The BTs show high reactivity, even though they cannot recreate all demonstrations.
One additional prerequisite for the behavior implementation of the Kobo robot was, to be able to add new demonstration data. This would facilitate to teach the robot even after behavior generation and to refine its behavior. With this approach, the addition of new demonstration data could possibly alter the preconditions and ramifications of the actions and the goal conditions. In this case, the behavior generation would have to restart from the beginning, possibly altering the initial BT and the atomic BTs for expansion. This means that this approach does not enable *online* learning and behavior refinement would take about the same computation time as for initial generation.

6.4 Automatic Action Sequence Creation and BT Specialization on Demonstration Data

The BTs created with this approach are characterized by many unnecessary nodes. They contain significantly more redundant nodes than with the PA-BT approach. This originates from the specialization approach. The initial BT is the biggest one and with every specialization, nodes are eliminated. Also, the proposed pruning algorithm of section 4.2.2 is able to eliminate further redundant nodes. For example, the generated BT for demonstration set Δ_1 consists of 36 nodes before (see figure 6.19) and 16 nodes after pruning. In fact, after pruning, the generated BT with this approach is also exactly the same as proposed by Colledanchise and Ögren [Colledanchise and Ögren, 2017].
The generated BT for Δ_2 consists of 34 nodes before pruning and 22 nodes afterwards. The BTs which are generated with this approach tend to form a flat structure, which originates from the sequential approach. The generated BT after pruning (seen in figure 6.20) models the inverted preconditions before every action. The action is then only executed, if no inverted precondition returns SUCCESS, i.e. every precondition is given. The BT specialized on Δ_3 consists of 42 nodes before and 22 nodes after pruning. The pruned result is equal to the pruned BT, generated on Δ_2 (see figure 6.20). Finally, the BT generated on Δ_4 consists of 44 nodes before and 24 nodes after pruning. In summary, 35% to 56% of created nodes have been found redundant with this approach. On average 46% of created nodes have been removed. This emphasizes again the need for pruning.
The BTs generated via BT specialization perform at least as good on the demonstration data, as the BTs generated via PA-BT (see table 6.3). The BTs generated on Δ_1 via PA-BT and BT specialization are equal after pruning (see figure 6.15) and therefore perform equally. The BTs generated on Δ_2 however are not equal (see figure 6.16 and figure 6.20) but seem

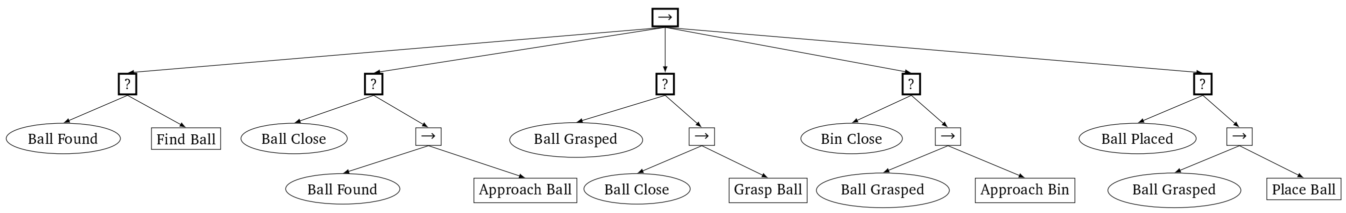


Figure 6.7.: Approach 1: PA-BT with multiple goal conditions: Generation 5. The BT of the previous generation, seen in figure 6.6 has been expanded. The Condition node *Ball Placed* was replaced with its corresponding atomic BT

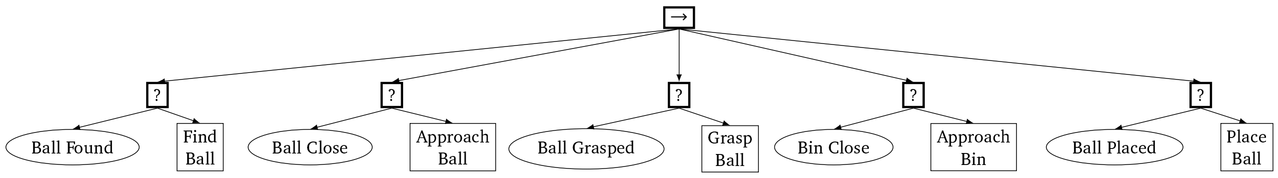


Figure 6.8.: Approach 1: PA-BT with multiple goal conditions: Generation 5, seen in figure 6.6, after pruning.

to perform equally on the demonstration data. With this approach as well, the BT expansion on Δ_2 and Δ_5 are exactly the same and therefore perform equally. The BT specialization on Δ_2 and Δ_5 performs better than the PA-BT approach. It succeeds on 26 of 32 demonstrations of Δ_1, which is rather interesting because for the PA-BT approach, the same goal conditions were detected, but it only succeeded on 11 of 32 demonstrations. It is also to note that the underlying behavior of demonstration Δ_1 is different from the behavior modeled in Δ_3, which is a subset of Δ_2. Furthermore, the generated BT, based on Δ_3 performs successfully on all demonstrations of Δ_2, whereas the equivalent BT generated using PA-BT only succeeded on 13 of 34 demonstrations. At closer inspection, the BT expanded on Δ_3 is also exactly equal to the BTs based on Δ_2 and Δ_5. It seems that the restriction on only sequential, loop-free behavior expanded the capability to generate a useable BT with incomplete demonstration data. The BT generated from Δ_4, which only consists of 2 demonstrations, performs successfully on 29 of 34 demonstrations, whereas the BT created using PA-BT on this demonstration only succeeded on 8 out of 34 demonstrations. The requirement for demonstration data to contain an inherent topological order of performed actions already sets the overall structure of the BT and therefore simplifies the range of possibilities. The BTs based on Δ_2, Δ_3 and Δ_5 perform successfully on 100% of demonstrations with external factors. The BT generated from Δ_1 with this approach succeeds equally on Δ_5 as its equivalent created with PA-BT. In contrast, the BT based on Δ_4, which performed better on Δ_2 than its PA-BT equivalent, also performs very well with added external factors. Out of 9 demonstrations with external actions, 6 could be recreated.
Because this approach cannot deal with loops in the given action sequences, it is rather limited. No topological order can be found if actions are executed more than once. Also, it only works if a coherent order exists in the given action sequences. If the order, in which two actions are executed, can be changed and if this change is present in the demonstration data, no topological ordering of actions can be found.
In contrast to the approach using PA-BT, this specialization approach would be able to add new demonstration data to an already existing BT, without reasoning over the full set of demonstration data. The specialization algorithm could simply run on the new demonstration and check if the given action sequence is executed by the BT. If not, the specialization step as proposed in section 4.3.2 is executed until the BT also recreates the new demonstration data. This *online* learning capability could be used to easily correct an already existing behavior. The demonstrator could interrupt the execution and show the robot the correct execution. The algorithm could then continue on the current BT with the new demonstration data to possibly refine the behavior.

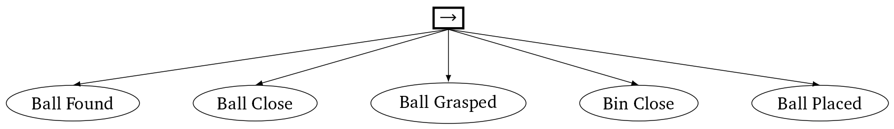


Figure 6.9.: Approach 2: PA-BT with automatic action detection on demonstration data Δ_1: Generation 0.

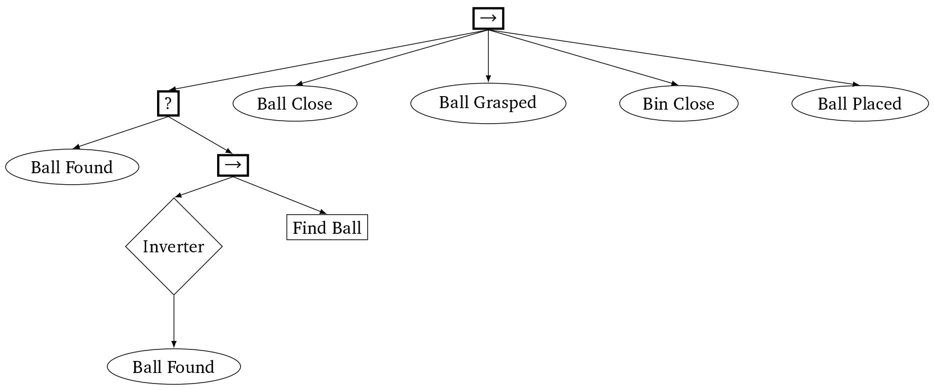


Figure 6.10.: Approach 2: PA-BT with automatic action detection on demonstration data Δ_1: Generation 1.

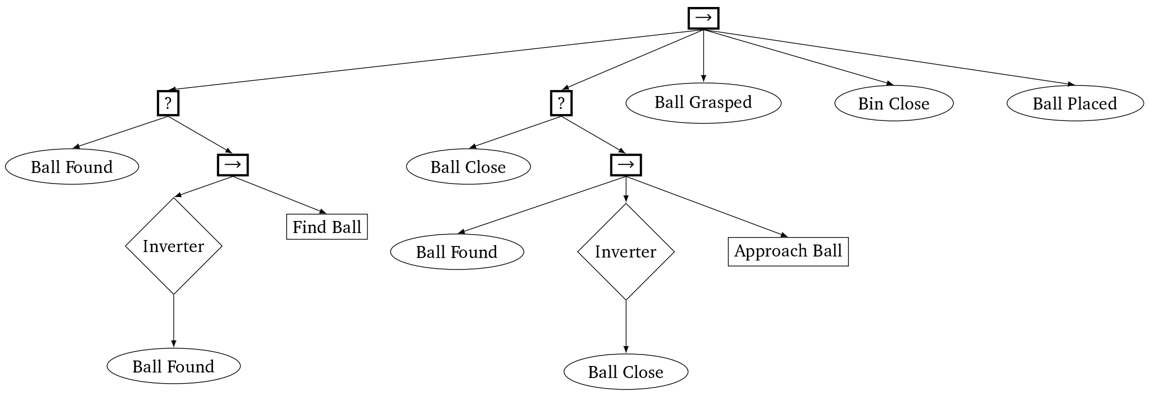


Figure 6.11.: Approach 2: PA-BT with automatic action detection on demonstration data Δ_1: Generation 2.

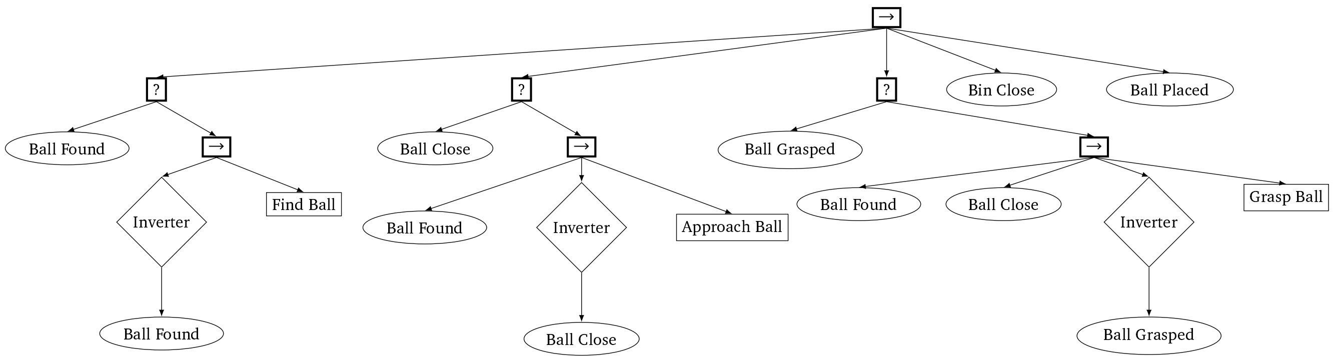


Figure 6.12.: Approach 2: PA-BT with automatic action detection on demonstration data Δ_1: Generation 3.

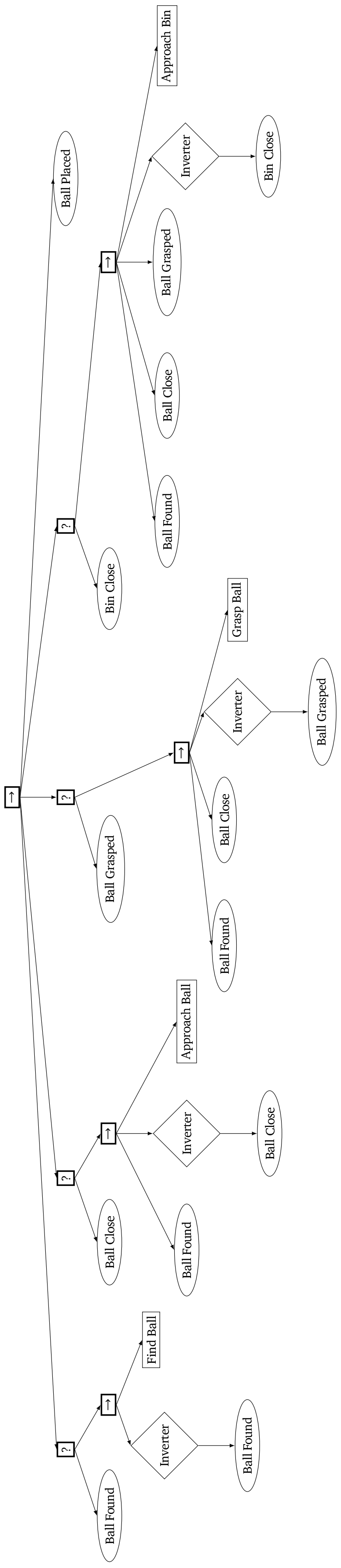


Figure 6.13.: Approach 2: PA-BT with automatic action detection on demonstration data Δ_1: Generation 4.

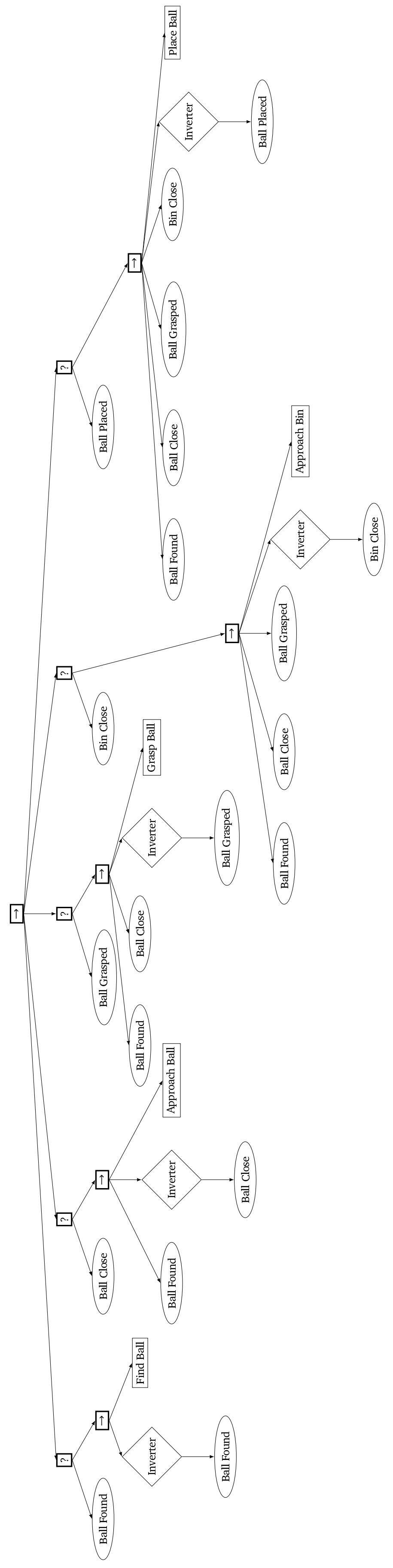


Figure 6.14.: Approach 2: PA-BT with automatic action detection on demonstration data Δ_1: Generation 5.

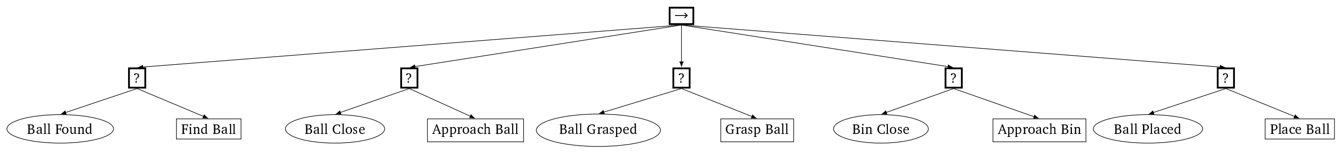


Figure 6.15.: Approach 2: PA-BT with automatic action detection on demonstration data Δ_1: Generation 5, seen in figure 6.14, after pruning.

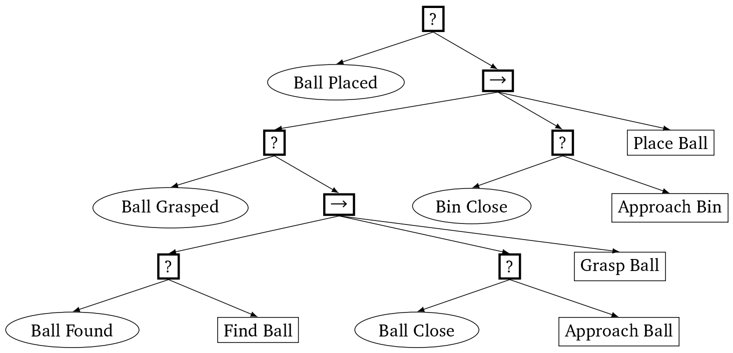


Figure 6.16.: Approach 2: PA-BT with automatic action detection on demonstration data Δ_2 : Generation 5 after pruning. The same BT is created with PA-BT with automatic action detection on Δ_5

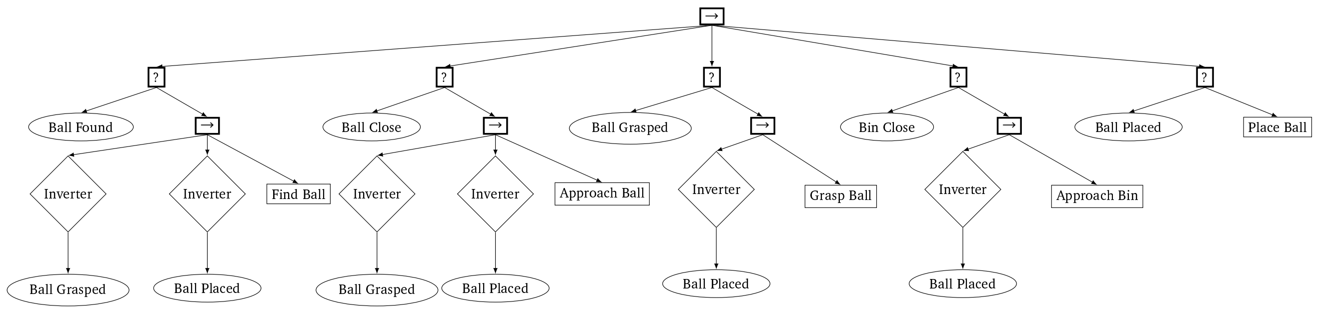


Figure 6.17.: Approach 2: PA-BT with automatic action detection on demonstration data Δ_3 : Generation 5 after pruning.

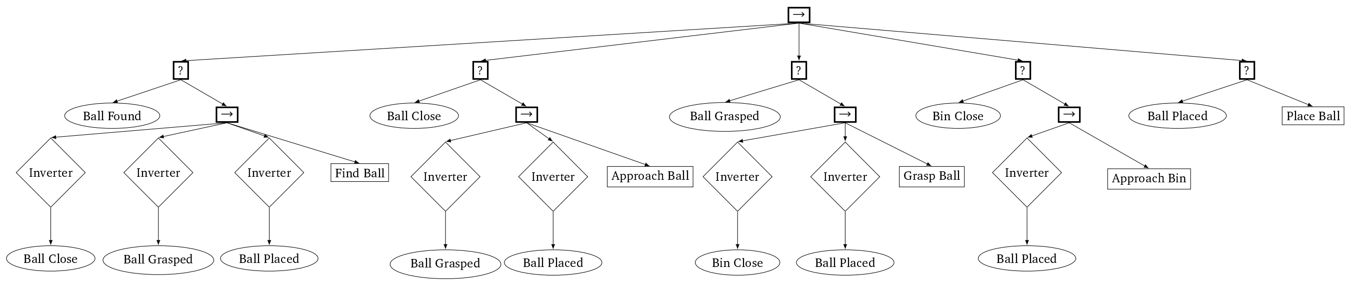


Figure 6.18.: Approach 2: PA-BT with automatic action detection on demonstration data Δ_4 : Generation 5 after pruning.

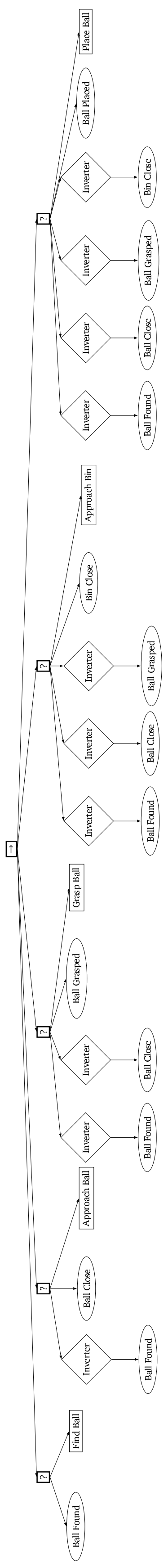


Figure 6.19.: Approach 3: Automatic action sequence creation and BT specialization on demonstration data Δ_1: Generation 14 before pruning. Pruning of this BT results in the proposed BT by [Colledanchise and Ögren, 2017] also seen in figure 6.15.

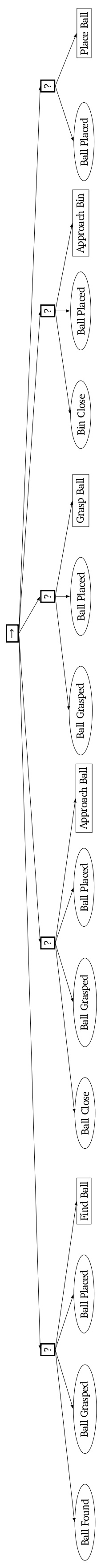


Figure 6.20.: Approach 3: Automatic action sequence creation and BT specialization on demonstration data Δ_2: Generation 11 after pruning and on demonstration data Δ_3: Generation 9 after pruning.

Tested on / *Generated from*	Δ_1	Δ_2	Δ_3	Δ_4	Δ_5
Δ_1	32/32 (100%)	13/34 (≈38%)	8/8 (100%)	2/2 (100%)	21/43 (≈49%)
Δ_2	26/32 (≈81%)	34/34 (100%)	8/8 (100%)	2/2 (100%)	43/43 (100%)
Δ_3	26/32 (≈81%)	34/34 (100%)	8/8 (100%)	2/2 (100%)	43/43 (100%)
Δ_4	21/32 (≈66%)	29/34 (≈85%)	5/8 (≈63%)	2/2 (100%)	35/43 (≈81%)
Δ_5	26/32 (≈81%)	34/34 (100%)	8/8 (100%)	2/2 (100%)	43/43 (100%)

Table 6.3.: The results of the sanity checks on different demonstrations for the BTs generated via BT specialization. Every result represents how many demonstrations were tested successfully divided by the amount of demonstrations the BT was tested on.

7 Conclusion & Outlook

In this thesis, we compared the possibilities of FSMs and BTs to model high-level robot behavior in the context of the KoBo34 project. First, we defined the concepts of FSMs and BTs and then concentrated on the possibility to generate them automatically from demonstration data. We showed the advantages and disadvantages of both behavioral representations on the basis of four use cases defined by the Kobo34 project and explained our decision to continue to work with BTs. The main advantage of BTs over FSM is their reactive behavior, without the addition of further nodes. However, BTs need another mindset than for FSMs and the BT tools are less mature than the software available for FSMs. Then, we presented three approaches to automatically generate BTs from demonstration. The implementation is based on the *behaviortree.cpp* framework which has been implemented for the use with the *Robot Operating System* (ROS).

The first approach is based on *Planning and Acting on using Behavior Trees* (PA-BT), which generates atomic BTs and therefore requires the definition of goal conditions and a set of actions with preconditions and ramifications. To further investigate if this information could be given by a caregiver or elderly person, a study has been performed. This study showed that participants intuitively concentrated on action sequences and that, especially for older people, the concept of preconditions and ramifications had to be clarified multiple times. We generated working BTs from 4 out of 8 action definitions given by participants but also showed that this approach was more difficult than an action sequence approach and that it was even impossible to define actions in this manner for some participants.

We therefore developed, for our second approach, a simple algorithm to determine preconditions and ramifications of all actions from demonstration data. This set of actions was then also used to generate BTs via the PA-BT approach. Five sets of demonstrations were created with different state space coverage and also including external actions. Experiments showed that BTs generated with this approach can always recreate the behavior of the demonstration data, their expansion was based on. However, the generated BTs contain a high amount of redundant nodes, which can be pruned by our proposed pruning algorithm. On average, this algorithm removed 50% of the created nodes and therefore increased readability. The generated BTs show reactive behavior and are capable to even react to external actions, although the demonstrations, their expansion was based on, contained none. Furthermore, the lack of demonstration data can lead to unintended goal conditions. We implemented the possibility to add new demonstration data after the expansion of a BT, but this approach is not capable of *online* learning.

The third approach is inspired by the findings of the study and the lack of online learning capabilities of previous approaches. We developed a way of BT expansion, based on action sequences, which was the first intuitive approach of all study participants. This approach analyzes the topological order of actions sequences from demonstration and generates an overly general BT, which is then specialized on demonstration data. The resulting BTs can also completely recreate the behavior found in the demonstration data they are based on, but also contain a high amount of redundancy, which makes pruning necessary as well. For this approach, the pruning algorithm removed 46% of created nodes on average. Furthermore, we found that BTs generated with this approach can even outperform BTs expanded with the second approach on incomplete demonstration sets. They also show the same reactive behavior as the BTs expanded via PA-BT. However, this approach is more limited than the second approach, because it cannot deal with loops in the action sequences or actions, which can be executed in different orders.

This thesis could be the basis for some future work. First of all, the current implementation, which is based on two *SimpleActionServers*, could be improved. Unfortunately, the communication between the tree nodes and the State Server currently takes about one second per node evaluation. Therefore, the execution of a complex BT can be very slow. The reason for this could not be found over the course of this thesis, but we think that it is necessary to improve the speed of the implementation for more complex scenarios. In that regard, the thesis mostly concentrated on one simple pick and place task. This task was chosen for its simplicity but does not represent a real use case of the KoBo34 project. Future demonstration data could be used to generate and test more appropriate behaviors. Furthermore, the demonstration data for this thesis was created by hand. A way of recording demonstrations will be necessary to test the behavior generation with non-programmers (i.e. caregivers and elderly people).

The proposed second approach, based on automatic action detection, currently uses a very simple algorithm to detect the preconditions and ramifications of all actions. There is certainly the possibility to implement different, more complex ways of action detection. Future implementations could be based on the work of Amir and Chang [Amir and Chang, 2008] or Čertickỳ [Čertickỳ, 2014]. The topological ordering, used for the third approach, limits the possibilities of this approach and could be expanded to deal with alternative action orders and loops. All generated BTs contained unnecessary nodes, which we have been able to detect and remove with the proposed pruning algorithm. This algorithm could be refined to further simplify the generated BTs and therefore to increase readability.

Finally, the study we performed only consists of 8 participants. A study with more participants could certainly investigate the possibilities of robot teaching in more detail. It could either concentrate on elderly people or try to show if there is a relation between the age of participants and the success rate of the BTs generated from their answers. Furthermore, the question of readability could be investigated. For this thesis, the readability of a behavioral structure was a very vague prerequisite, which was treated as equivalent to the complexity of a BT or FSM (i.e. the number of nodes or states and transitions). However, BTs demand another mindset as FSMs. A future study could directly compare the readability of these behavioral representations and possibly develop a clearer measure for this currently very vague term. Finally, a future study could also expand the robot teaching from a verbal approach to a Learning from Demonstration approach and investigate the generated results and the reaction of the elderly people to this kind of robot teaching.

Bibliography

[Adadi and Berrada, 2018] Adadi, A. and Berrada, M. (2018). Peeking inside the black-box: A survey on explainable artificial intelligence (xai). *IEEE Access*, 6:52138–52160.

[Allmendinger and Ebner, 2006] Allmendinger, J. and Ebner, C. (2006). Arbeitsmarkt und demografischer Wandel: Die Zukunft der Beschäftigung in Deutschland. *Zeitschrift für Arbeits-und Organisationspsychologie A&O*, 50(4):227–239.

[Amir and Chang, 2008] Amir, E. and Chang, A. (2008). Learning partially observable deterministic action models. *Journal of Artificial Intelligence Research*, 33:349–402.

[Argall et al., 2009] Argall, B. D., Chernova, S., Veloso, M., and Browning, B. (2009). A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483.

[Bagnell et al., 2012] Bagnell, J. A., Cavalcanti, F., Cui, L., Galluzzo, T., Hebert, M., Kazemi, M., Klingensmith, M., Libby, J., Liu, T. Y., Pollard, N., et al. (2012). An integrated system for autonomous robotics manipulation. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2955–2962. IEEE.

[Bundesagentur für Arbeit, 2019] Bundesagentur für Arbeit (2019). Arbeitsmarktsituation im Pflegebereich. *Berichte: Blickpunkt Arbeitsmarkt*. `https://statistik.arbeitsagentur.de/Statischer-Content/Arbeitsmarktberichte/Berufe/generische-Publikationen/Altenpflege.pdf`. Last accessed on Sep 23, 2019.

[Čertickỳ, 2014] Čertickỳ, M. (2014). Real-time action model learning with online algorithm 3 sg. *Applied Artificial Intelligence*, 28(7):690–711.

[Colledanchise et al., 2016] Colledanchise, M., Almeida, D., and Ögren, P. (2016). Towards blended reactive planning and acting using behavior trees. *arXiv preprint arXiv:1611.00230*.

[Colledanchise and Ögren, 2017] Colledanchise, M. and Ögren, P. (2017). Behavior trees in robotics and ai: an introduction. *arXiv preprint arXiv:1709.00084*.

[Doblhammer and Ziegler, 2006] Doblhammer, G. and Ziegler, U. (2006). Future elderly living conditions in europe: demographic insights. In *Gender, Health and Ageing*, pages 267–292. Springer.

[Dorsch - Lexikon der Psychologie, 2019] Dorsch - Lexikon der Psychologie (2019). Hohes Alter - Dorsch Lexikon der Psychologie - Verlag Hans Huber. Last accessed on Sep 23, 2019.

[Girault et al., 1999] Girault, A., Lee, B., and Lee, E. A. (1999). Hierarchical finite state machines with multiple concurrency models. *IEEE Transactions on computer-aided design of integrated circuits and systems*, 18(6):742–760.

[Grollman and Jenkins, 2008] Grollman, D. H. and Jenkins, O. C. (2008). Sparse incremental learning for interactive robot control policy estimation. In *2008 IEEE International Conference on Robotics and Automation*, pages 3315–3320. IEEE.

[Grollman and Jenkins, 2010a] Grollman, D. H. and Jenkins, O. C. (2010a). Can we learn finite state machine robot controllers from interactive demonstration? In *From Motor Learning to Interaction Learning in Robots*, pages 407–430. Springer.

[Grollman and Jenkins, 2010b] Grollman, D. H. and Jenkins, O. C. (2010b). Incremental learning of subtasks from unsegmented demonstration. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 261–266. IEEE.

[Harel, 1987] Harel, D. (1987). Statecharts: A visual formalism for complex systems. *Science of computer programming*, 8(3):231–274.

[Hopcroft et al., 2014] Hopcroft, J. E., Motwani, R., and Ullman, J. D. (2014). Introduction to automata theory, languages and computation.

[Isla, 2005] Isla, D. (2005). Gdc 2005 proceeding: Handling complexity in the halo 2 ai. `https://www.gamasutra.com/view/feature/130663/gdc_2005_proceeding_handling_.php`. Last accessed on Jul 04, 2019.

[Kress-Gazit et al., 2011] Kress-Gazit, H., Wongpiromsarn, T., and Topcu, U. (2011). Correct, reactive, high-level robot control. *IEEE Robotics & Automation Magazine*, 18(3):65–74.

[Maniatopoulos et al., 2016] Maniatopoulos, S., Schillinger, P., Pong, V., Conner, D. C., and Kress-Gazit, H. (2016). Reactive high-level behavior synthesis for an atlas humanoid robot. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4192–4199. IEEE.

[Niekum et al., 2015] Niekum, S., Osentoski, S., Konidaris, G., Chitta, S., Marthi, B., and Barto, A. G. (2015). Learning grounded finite-state representations from unstructured demonstrations. *The International Journal of Robotics Research*, 34(2):131–157.

[Paxton et al., 2017] Paxton, C., Hundt, A., Jonathan, F., Guerin, K., and Hager, G. D. (2017). Costar: Instructing collaborative robots with behavior trees and vision. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 564–571. IEEE.

[Piterman et al., 2006] Piterman, N., Pnueli, A., and Sa'ar, Y. (2006). Synthesis of reactive (1) designs. In *International Workshop on Verification, Model Checking, and Abstract Interpretation*, pages 364–380. Springer.

[Quigley et al., 2009] Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., and Ng, A. (2009). Ros: an open-source robot operating system. In *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics*, Kobe, Japan.

[Quinlan, 2014] Quinlan, J. R. (2014). *C4. 5: programs for machine learning*. Elsevier.

[Robertson and Watson, 2015] Robertson, G. and Watson, I. (2015). Building behavior trees from observations in real-time strategy games. In *2015 International Symposium on Innovations in Intelligent SysTems and Applications (INISTA)*, pages 1–7. IEEE.

[Sagredo-Olivenza et al., 2017] Sagredo-Olivenza, I., Gómez-Martín, P. P., Gómez-Martín, M. A., and González-Calero, P. A. (2017). Trained behavior trees: Programming by demonstration to support ai game designers. *IEEE Transactions on Games*.

[Schwab and Hlavacs, 2015] Schwab, P. and Hlavacs, H. (2015). Capturing the essence: Towards the automated generation of transparent behavior models. In *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*.

[Wagner et al., 2006] Wagner, F., Schmuki, R., Wagner, T., and Wolstenholme, P. (2006). *Modeling software with finite state machines: a practical approach*. Auerbach Publications.

[Weber et al., 2011] Weber, B. G., Mateas, M., and Jhala, A. (2011). Building human-level ai for real-time strategy games. In *2011 AAAI Fall Symposium Series*.

[Weber et al., 2010] Weber, B. G., Mawhorter, P., Mateas, M., and Jhala, A. (2010). Reactive planning idioms for multi-scale game ai. In *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*, pages 115–122. IEEE.

A Appendix

A.1 Questionnaire for demonstration evaluation

The following study was done with 8 people in a range from 21 to 89 years old (52 years on average). The participants were asked for their age to find possible relations with the kind of answers given. The German language was chosen to ensure the possibility to work with elderly people. The headlines of the following subsections were not visible to the participants and the individual sections were presented one by one.

A.1.1 Free description of behavior

The first question is intentionally a very open question. This question is asked first, to see the intuitive way of a person to describe the behavior. Only a symbolic illustration is given (see figure A.1). Without giving too much information, it is avoided to influence the answer to the following question: You want to teach a robot to place a ball in a bin. How would you describe this to it? After the description is given, two other questions are posed to find out how difficult this kind of description was and if there was a desire to know more about the robot: How difficult was this kind of description for you (1 = very easy, 5 = very hard)? Would you have liked to have any further information about the robot for this task?

Sie möchten einem Roboter beibringen, wie er einen Ball in einen Eimer befördert. Wie würden Sie ihm das erklären?

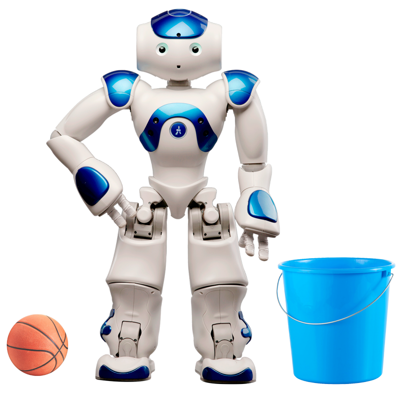


Figure A.1.: Illustration for the performed study. This illustration shows all elements of the given example to participants, which should inspire them to describe the robot behavior for placing a ball in a bin.

Wie leicht ist Ihnen diese Art der Beschreibung gefallen (1 = sehr leicht, 5 = sehr schwer)? Hätten Sie sich für diese Aufgabe zusätzlichen Informationen über den Roboter gewünscht?

A.1.2 Free description with given action set and conditions

For the second question, the possible actions (Approach Ball, Place Ball, Approach Bin, Find Ball and Grasp Ball) and conditions (Ball Grasped, Bin Close, Ball Placed, Ball Found, Ball Close) representing the state are given. The question is, if the behavior would be described differently with these informations. After this question, it is asked again, how difficult this kind of description was and if there was any information missing about the actions and conditions.

Dem Roboter stehen folgende Aktionen zur Verfügung:

- Zum Ball gehen
- Ball in den Eimer legen
- Zum Eimer gehen

- Ball finden
- Ball greifen

Zusätzlich stehen dem Roboter folgende Beobachtungen zur Verfügung:

- Hat der Roboter den Ball gegriffen
- Ist der Eimer in der Nähe
- Liegt der Ball im Eimer
- Wo ist der Ball
- Ist der Ball in der Nähe

Wie würden Sie ihm die Aufgabe nun schildern?
Wie leicht ist Ihnen diese Art der Beschreibung gefallen (1 = sehr leicht, 5 = sehr schwer)? Gibt es Informationen über die Aktionen und Beobachtungen des Roboters, die Ihnen für diese Beschreibung eventuell gefehlt haben?

A.1.3 Description of preconditions, ramifications and goal conditions

The last question is asked to compare how preconditions and ramifications for actions are defined. The exact question is: Try to give exactly the preconditions which have to be true for the execution of an action. And also give the ramifications of the action, the robot can observe. The answers given have been filled in a table like table A.1 and it was clarified that only the five conditions (Approach Ball, Place Ball, Approach Bin, Find Ball and Grasp Ball) presented before can be ramification or precondition. Two follow-up questions are posed. The first, how difficult this kind of description was, and the second, to find out how many questions of the robot would be tolerable for a user. The exact translation: How many questions of the robot would you have accepted for this task?

Versuchen Sie zu jeder Aktion genau anzugeben, welche Voraussetzungen für die Ausführung der Aktion erfüllt sein müssen. Und geben Sie dazu an, welche Auswirkungen diese Aktion auf die Beobachtungen des Roboters hat.

	Voraussetzungen	**Auswirkungen**
Ball finden		
Zum Ball gehen		
Ball greifen		
Zum Eimer gehen		
Ball in den Eimer legen		

Table A.1.: Table of preconditions and ramifications for all actions to fill in. Additionally, all conditions the robot can observe, have been presented.

Wie leicht ist Ihnen diese Art der Beschreibung gefallen (1 = sehr leicht, 5 = sehr schwer)? Wie viele Fragen des Roboters würden Sie für solch eine Aufgabe akzeptieren?

A.2 Study results

The following notes have been taken while interviewing the study participants and according to an audio recording. The recording is not part of the appendix for data protection reasons.

A.2.1 Participant 1 (Age 60)

The following notes have been taken for participant 1.

Free description of behavior

- "Strecke deine rechte Hand nach unten. Mach deine Hand auf. Schließe deine Hand wieder, dass du den Ball festhältst. Nach oben heben und drehe dich in Richtung vom Eimer, mache deine Hand wieder auf und lasse den Ball fallen."
- Movements have been demonstrated physically: the arm movement of the robot, grasping, turning to the bin and placing the ball.
- Difficulty level of this description: 1
- Further information needed: no
- Time taken for this description: 43 seconds.

Free description with given action set and conditions

- Similar description given as for first question: "Arm ausstrecken, Arm nach unten, greifen, Arm nach oben, Richtung Eimer, fallen lassen."
- After explaining the actions in detail, long pauses of reflection.
- Description was not changed. Even after explaining, that the robot has no direct action to stretch out its arm.
- The Participant responded positive, when he was asked, if he thought that the behavior would work like that.
- Difficulty level of this description: 2
- Further information needed:
  - can the robot walk?
  - can the robot see the ball?
  - can the robot turn?
  - can the robot sense if it has grasped the ball?
  - how does the robot know that it has to move its arm to grasp the ball?
  - can the robot know how strong it grasps the ball?
- Time taken for this description: 3 minutes, 34 seconds.

Description of preconditions, ramifications and goal conditions

- The results of this description can be seen in table A.2.
- No BT could be generated from this description, because of conflicting preconditions and ramifications (for example *BallFound* as precondition and ramification for the action *Find Ball*)
- Even after clarifying that only given conditions can be preconditions or ramifications, the first answer for preconditions of *Find Ball* was: "Also, er muss seinen Arm bewegen." and "Dass er sich bewegen kann, der Roboter".

- Further clarification for the meaning of *precondition* and *ramification* was needed. Especially the temporal order (i.e. preconditions before and ramifications after the action)
- Actions have been mixed up: while describing the preconditions and ramifications for action *Find Ball*, walking to the ball was described already.
- Difficulty level of this description: 3
- Acceptable number of questions for this task: 4-5
- Time taken for this description: 14 minutes, 18 seconds.

	Preconditions	**Ramifications**
Find Ball	BallFound, BallClose	BallFound
Approach Ball	BallFound,BallClose	BallClose
Grasp Ball	BallClose, BallFound	BallGrasped
Approach Bin	BinClose	BinClose
Place Ball	BallGrasped, BallFound, BinClose	BallPlaced, BallFound, BallGrasped

Table A.2.: Table of preconditions and ramifications filled in by participant 1.

A.2.2 Participant 2 (Age 89)

The following notes have been taken for participant 2.

Free description of behavior

- "Ich weiß doch nicht, wie der sich bewegen kann. Kann der sich bücken?"
- "Bücken, mit zwei Händen den Ball fassen und dann wieder auf und in den Eimer schmeißen."
- Difficulty level of this description: 2
- Further information needed: movement possibilities of the robot
- Time taken for this description: 25 seconds.

Free description with given action set and conditions

- "Schau mal, wo der Eimer ist. Und dann schaue, ob der Ball in deiner Nähe ist. Dann kannst du den Ball reinwerfen."
- Difficulty level of this description: 3
- Further information needed: can the robot turn?
- Time taken for this description: 1 minutes, 42 seconds.

Description of preconditions, ramifications and goal conditions

- The results of this description can be seen in table A.3.
- No BT could be generated from this description, because of conflicting preconditions and ramifications (for example *Bin Close* as precondition and ramification for the action *Approach Bin*)
- The questions were found as cumbersome: “Das ist so ein Umstand.”
- For the ramifications of *FindBall*, other actions have been described already: “Dann muss er sich in die Richtung drehen. Und dann muss er sich bücken.”
- Even when asked directly for the observations of the robot, the participant concentrated on actions: “Dann muss er sich bücken.”
- “Liegt der Ball im Eimer? Und wenn nicht, dann muss er ihn reintun.”
- While discussing the ramifications for the action *Grasp Ball*, the condition of *Ball Grasped* was not mentioned. When clarifying this ramification, the condition has been mentioned as *obvious*: “Ich habe gedacht, das wäre selbstverständlich.”
- Proposed precondition for the action *Place Ball*: “Ja, der braucht bewegliche Hände.”
- The negation of *Ball Placed* is proposed as precondition for the action *Place Ball*.
- Difficulty level of this description: 3
- Acceptable number of questions for this task: 5
- Time taken for this description: 11 minutes, 6 seconds.

	Preconditions	**Ramifications**
Find Ball	BallFound	
Approach Ball	BallFound	
Grasp Ball	BallClose, BallFound	(BallGrasped)
Approach Bin	BinClose	BinClose
Place Ball	BallGrasped, BallFound, BinClose, ¬BallPlaced	BallFound, BallClose, BallGrasped, BinClose, BallPlaced

Table A.3.: Table of preconditions and ramifications filled in by participant 2.

A.2.3 Participant 3 (Age 62)

The following notes have been taken for participant 3.

Free description of behavior

- Only two actions have been given. “Grundsätzlich, Ball greifen und dann Ball im Eimer ablegen.”
- “Wenn das so möglich ist, das als Befehl zu geben.”
- Difficulty level of this description: 1
- Further information needed:
  - movement possibilities of the robot
  - which functions does the robot have
- Time taken for this description: 41 seconds.

Free description with given action set and conditions

- “Zuerst wäre Ball finden, dann zum Ball gehen, dann den Ball greifen, zum Eimer gehen, in den Eimer legen und dann noch prüfen, ob der Ball im Eimer liegt.”
- “Vergessen habe ich jetzt, ob der Eimer in der Nähe ist.”
- Difficulty level of this description: 2
- Further information needed: no
- Time taken for this description: 1 minutes, 2 seconds.

Description of preconditions, ramifications and goal conditions

- The results of this description can be seen in table A.4.
- No BT could be generated from this description, because of missing ramifications (for example for the action *Approach Ball*)
- Further conditions were given: “Es hat sich verändert, dass er jetzt schauen muss, ob der Eimer in der Nähe ist.”
- Actions were mixed up: “Weil, wenn er den Ball hat, er hat den Ball entdeckt, geht er dann denke ich zum Ball, und greift ihn auch. Oder ist das dann schon die nächste Aktion?”
- Limits were questioned: “Das ist jetzt zum Beispiel, wenn der Ball jetzt auf dem Regal oben liegen würde, wo er ihn nicht greifen kann...”
- Further preconditions for *Approach Ball* were given: “Dass der Weg dorthin für ihn frei ist.”
- Preconditions were mixed up with the preconditions of the action *Find Ball*: “In der Nähe haben wir ja schon gesagt.”, “Na, das haben wir ja da oben schon geprüft.”
- The definition of “Close” was unclear to the participant and had to be clarified further.
- Finally, no preconditions and ramifications for *Approach Ball* were given.
- The temporal order of preconditions and ramifications had to be clarified.
- “Er muss nah genug am Ball sein, dass er ihn greifen kann”. After asking if there was a condition modeling this: “Ja, vielleicht Ist der Ball in der Nähe. Aber das haben wir ja da oben schon geklärt.”
- The definition of “Close” had to be clarified again. “Close” and “Found” were mixed up.
- The separation of actions had to be explained further: “Ach so, also das heißt, bevor ich zum Eimer gehe, muss ich wieder prüfen, hat er jetzt die letzte Funktion ausgeführt, hat er den Ball richtig gegriffen.”
- When asking for a ramification of *ApproachBall*: “Er kann zum Eimer laufen.” Temporal order of preconditions and ramifications had to be clarified again. “Er kann jetzt den Ball in den Eimer legen.”
- Had to be clarified that actions are always successful.
- Separation of actions was mixed up again: “Voraussetzung ist, dass der Eimer dann in der Nähe ist. Aber er steht ja dort, steht am Eimer haben wir ja gesagt.”
- The ramification $\neg$*BallGrasped* for the action *PlaceBall* was given as well: “Wenn du das jetzt alles so genau, ausführlich machen willst.”
- When asked for the difficulty level: “Das war jetzt schon ein bisschen schwierig. Ich hab das wahrscheinlich alles ein bisschen falsch verstanden.”
- Difficulty level of this description: 4
- Acceptable number of questions for this task: 3

- Time taken for this description: 18 minutes, 15 seconds.

	Preconditions	Ramifications
Find Ball	BallFound, BallClose	BallFound
Approach Ball		
Grasp Ball		BallGrasped
Approach Bin	BallGrasped, ¬BinClose	BinClose
Place Ball	BinClose, BallGrasped	BallPlaced, ¬BallGrasped

Table A.4.: Table of preconditions and ramifications filled in by participant 3.

A.2.4 Participant 4 (Age 30)

The following notes have been taken for participant 4.

Free description of behavior

- An action sequence has been given. "Er soll auf dem Boden nach einem Ball suchen. Und wenn er ihn findet, soll er den Ball mit seiner Hand greifen und dann sich Richtung Eimer drehen und den Ball in den Eimer werfen."
- "Also, er schaut erst mal ob er einen Ball gegriffen hat oder nicht. Und wenn nicht, sollte er schauen, ob der Ball in der Nähe ist. Und dann kann er zum Ball gehen. Und dann kann er den Ball greifen. Und dann muss er schauen, ob der Eimer in der Nähe ist. Dann kann er zum Eimer gehen und den Ball in den Eimer legen . "
- Difficulty level of this description: 2
- Further information needed:
  - does the robot know what a ball/bin is
  - how far can the robot look for the ball
  - if he can grasp the ball
  - how does the robot know that this bin is the right bin
- Time taken for this description: 27 seconds.

Free description with given action set and conditions

- Difficulty level of this description: 3
- Further information needed:
  - definition of "Close"
  - do actions always succeed
- Time taken for this description: 1 minutes, 12 seconds.

Description of preconditions, ramifications and goal conditions

- The results of this description can be seen in table A.5.
- A BT, expanded from the given action set can be seen in figure A.2

- When asked about the difficulty level of this description: “Ja, am Anfang musste ich einfach verstehen, wie das funktioniert. Und ab dem zweiten hatte ich dann verstanden.”
- Difficulty level of this description: 3
- Acceptable number of questions for this task: 2
- Time taken for this description: 4 minutes, 43 seconds.

	Preconditions	**Ramifications**
Find Ball	¬BallPlaced	BallFound
Approach Ball	BallFound	BallClose
Grasp Ball	BallClose	BallGrasped
Approach Bin	BallGrasped	BinClose
Place Ball	BinClose	BallPlaced

Table A.5.: Table of preconditions and ramifications filled in by participant 4.

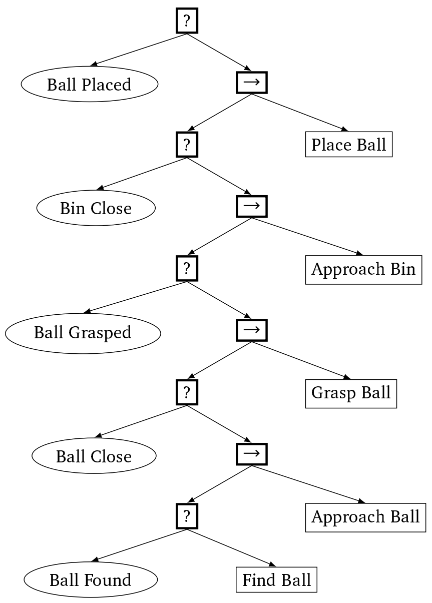


Figure A.2.: BT generated from the action set given by participant 4, seen in table A.5 with approach 1: PA-BT with given actions and goal condition *BallPlaced*.

A.2.5 Participant 5 (Age 27)

The following notes have been taken for participant 5.

Free description of behavior

- “Erstmal soll er sich vor den Ball stellen, dann sich bücken, mit den Armen den Ball in die Hände nehmen, sich wieder aufrichten und dann Richtung Eimer laufen. Und dann entweder den Ball reinfallen lassen oder am besten sich auch bücken und den Ball reinlegen.”
- Difficulty level of this description: 1
- Further information needed: no, “Das Bild hat mir gereicht.”
- Time taken for this description: 24 seconds.

Free description with given action set and conditions

- “Ich weiß nicht, mir fällt da jetzt irgendwie nicht ein, was man da noch hinzufügen könnte.”
- After clarifying the kind of description: “Okay, also Ball finden, zum Ball gehen, Ball greifen, zum Eimer gehen, Ball in den Eimer legen.”
- After explaining that observations can be used as well: “Langsam kapiere ich das auch”
- “Er muss erstmal feststellen, wo er ist. Und dann, er ist in der Nähe. Dann zum Ball gehen, Ball greifen. Dann macht er diese Beobachtung, dass er ihn jetzt gegriffen hat. Dann zum Eimer gehen und Ball in den Eimer legen. Und dann stellt er fest, dass er im Eimer ist”
- “Jetzt war es ja schon wahnsinnig ausführlich eigentlich. Also immer das nochmal zu hinterfragen.”
- Difficulty level of this description: 3
- Further information needed: “Er könnte sich natürlich immer fragen: Hab ich genug Energie?”
- Time taken for this description: 3 minutes, 38 seconds.

Description of preconditions, ramifications and goal conditions

- The results of this description can be seen in table A.6.
- A BT, expanded from the given action set can be seen in figure A.3
- “Weil ich irgendwie checken musste, wie das funktioniert, aber dann eigentlich, dann ist es ja einfach.”
- Difficulty level of this description: 3
- Acceptable number of questions for this task: 3
- Time taken for this description: 12 minutes, 3 seconds.

	Preconditions	**Ramifications**
Find Ball	¬BallFound	BallFound
Approach Ball	¬BallClose, BallFound	BallClose
Grasp Ball	BallClose, ¬BallGrasped, BallFound	BallGrasped
Approach Bin	¬BinClose	BinClose
Place Ball	BallGrasped, BinClose	BallPlaced

Table A.6.: Table of preconditions and ramifications filled in by participant 5.

A.2.6 Participant 6 (Age 71)

The following notes have been taken for participant 6.

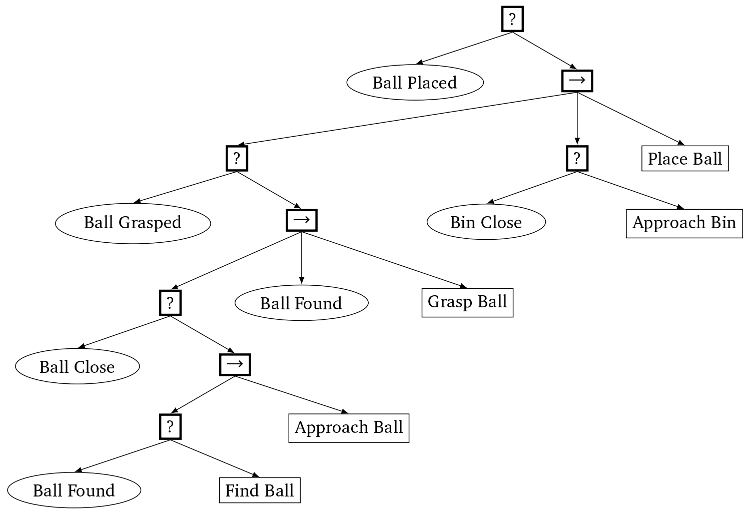


Figure A.3.: BT generated from the action set given by participant 5, seen in table A.6 with approach 1: PA-BT with given actions and goal condition *BallPlaced*.

Free description of behavior

- The participant tried to change the environment: "Also ich würde den Eimer vor den Roboter stellen."
- "Streck deinen rechten Arm aus, bück dich, nimm den Ball, mach eine leichte Wendung nach links und wirf ihn in den Eimer."
- When mentioned that the ball could be anywhere: "Dann muss ich dem Roboter sagen: Mach zwei Schritte, oder drei Schritte, je nach Entfernung."
- Difficulty level of this description: 1
- Further information needed: no
- Time taken for this description: 1 minute, 7 seconds.

Free description with given action set and conditions

- The participant was following the logical order of the actions with a finger: "Also erstmal müssen wir den Ball finden, dann gehen wir zum Ball hin, greifen den Ball, gehen zum Eimer, Ball in den Eimer legen."
- When asked, if the conditions were not relevant: "Also die braucht man jetzt nicht zu sortieren?"
- When asked, if the robot would understand a description without conditions, the participant was convinced that the behavior would work.
- Difficulty level of this description: 2
- Further information needed: no
- Time taken for this description: 1 minutes, 29 seconds.

Description of preconditions, ramifications and goal conditions

- The results of this description can be seen in table A.7.
- No BT could be generated from this description, because of missing ramifications (for example for the action *Find Ball*)

- The meaning of precondition and ramification had to be defined multiple times: “Um den Ball zu finden, da muss er erst mal wissen, wo der Ball ist. Und wenn er das weiß, dann findet er ja den.”
- The precondition *Ball Found* was given for the action *Find Ball*, but later erased: “Das da oben ist jetzt ein bisschen doof. Das passt nicht wirklich dahin. Ja findet er dann den Ball?”
- When comparing with all conditions: “Also von denen, passt hier nix hin”
- The precondition ¬BallGrasped was given for the action *GraspBall*: “Höchstens wäre hier, in dem Fall, indem ich es verneinend schreibe: Hat den Ball noch nicht gegriffen”
- The definition of “Close” was unclear to the participant and it had to be clarified that this means that the robot can interact with the object.
- This kind of description was found strange: “Ganz schön verquer zu denken.”
- Laughing: “Ich glaub, Programmierer wäre nichts für mich. Immer so verquer denken.”
- Difficulty level of this description: 3
- Acceptable number of questions for this task: 10
- Time taken for this description: 16 minutes, 55 seconds.

	Preconditions	**Ramifications**
Find Ball	(BallFound)	
Approach Ball	BallFound	BallClose
Grasp Ball	BallClose, ¬BallGrasped	BallGrasped
Approach Bin		BinClose
Place Ball	BallGrasped, BinClose, ¬BallPlaced	BallPlaced

Table A.7.: Table of preconditions and ramifications filled in by participant 6.

A.2.7 Participant 7 (Age 21)

The following notes have been taken for participant 7.

Free description of behavior

- “Also, der Roboter muss erstmal irgendwie wissen, wo der Ball ist. Und dann geht der dahin. Dann nimmt er ihn in die Hand und geht zum Eimer, wenn er weiß wo der Eimer ist. Und dann legt er ihn da rein. In den Eimer. Dann ist er fertig.”
- Difficulty level of this description: 2
- Further information needed: no, “Das hat mir jetzt irgendwie gereicht.”
- Time taken for this description: 21 seconds.

Free description with given action set and conditions

- The given list of actions and conditions inspired new questions: “Ich komm jetzt halt irgendwie da drauf, dass der Roboter ja vielleicht schon direkt den Ball gegriffen haben könnte. Und wenn man das weiter denkt, könnte er ja schon eigentlich ganz fertig sein.”

- The description was adapted: "Also ich hätte ihn vielleicht sozusagen erst mal gefragt, ob er irgendwie, vielleicht schon fertig ist oder den Ball schon hat. Ob er vielleicht schon beim Eimer ist. Und immer halt so schauen, ob die Sachen schon erledigt sind. Die man dann vielleicht gar nicht machen muss."
- "Immer so ein: Prüfen ob es schon gemacht ist, wenn nicht, machen. Und dann schrittweise durcharbeiten. Aber irgendwie rückwärts."
- When asked about the difficulty level: "Eher 4, weil es voll verwirrend ist."
- Difficulty level of this description: 4
- Further information needed: Could the condition *Ball Grasped* also influence other conditions like *Ball Found* or *Ball Close*
- Time taken for this description: 2 minutes, 6 seconds.

Description of preconditions, ramifications and goal conditions

- The results of this description can be seen in table A.8.
- A BT, expanded from the given action set can be seen in figure A.4
- The precondition ¬*BallPlaced* has been given but found as implicitly given by the precondition *BallGrasped*: "Weil, wenn der Ball in der Hand ist, kann er ja nicht im Eimer sein. Würde ich jetzt als Mensch sagen."
- The question if only one ball exists emerged: "Wenn wir sicher sein können, dass es nur einen Ball gibt, muss man es ihm auch nicht sagen. Man könnte irgendwie sagen: Brich ab, wenn der Ball in deiner Hand ist und im Eimer ist. Dann ruf um Hilfe."
- The ramification ¬*BallGrasped* for the action *PlaceBall* was given as well
- When asked for the difficulty level: "Also wenn man die Liste hier hat und die einfach sozusagen abarbeiten kann."
- "Schon viel Denkarbeit irgendwie."
- Difficulty level of this description: 3
- When asked for the acceptable number of questions for this task: "Ja, das ist irgendwie komplizierter als man denkt."
- "Bei 5 wäre ich schon irgendwie ein bisschen genervt."
- Acceptable number of questions for this task: 5
- Time taken for this description: 6 minutes, 28 seconds.

	Preconditions	**Ramifications**
Find Ball	¬BallGrasped, ¬BallPlaced, ¬BallFound	BallFound
Approach Ball	¬BallGrasped, ¬BallPlaced, BallFound, ¬BallClose	BallClose
Grasp Ball	¬BallGrasped, ¬BallPlaced, BallClose, BallFound	BallGrasped
Approach Bin	¬BallPlaced, BallGrasped, ¬BinClose	BinClose
Place Ball	BallGrasped, BinClose, (¬BallPlaced)	¬BallGrasped, BallPlaced

Table A.8.: Table of preconditions and ramifications filled in by participant 7.

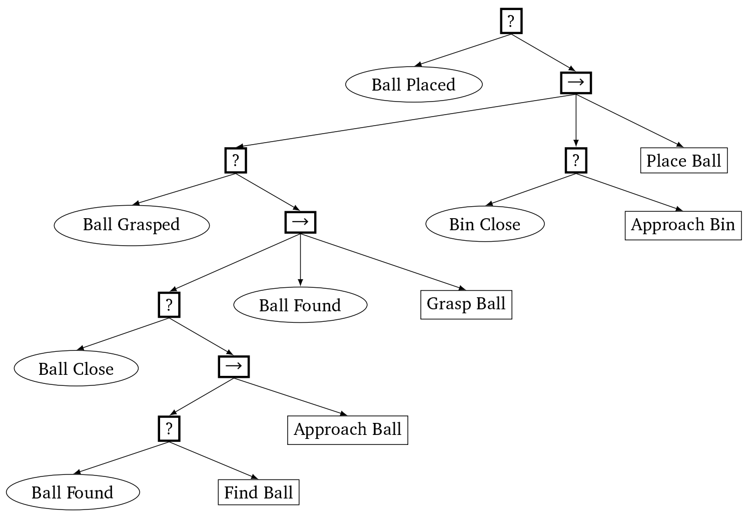


Figure A.4.: BT generated from the action set given by participant 7, seen in table A.8 with approach 1: PA-BT with given actions and goal condition *BallPlaced*.

A.2.8 Participant 8 (Age 56)

The following notes have been taken for participant 8.

Free description of behavior

- "Ball nehmen, von rechts nach links in den Eimer geben."
- After clarification that the position of the ball does not have to be as seen on the illustration: "Dann muss man den halt von einer anderen Stelle nehmen."
- "Den Ball nehmen, aufheben und in den Eimer geben."
- Difficulty level of this description: 1
- Further information needed: no
- Time taken for this description: 27 seconds.

Free description with given action set and conditions

- "Erstmal Ball finden, dann ist der Eimer in der Nähe, weil den muss er ja erst abchecken. (...) Dann als nächstes zum Ball gehen, Ball greifen, dann kontrollieren, hat er den Ball gegriffen. Dann zum Eimer gehen. Ball in den Eimer legen. Liegt der Ball im Eimer und wo ist der Ball, also ist der Ball drin."
- "Der muss natürlich wissen, wie er seine Gliedmaßen (...), wie er das alles benutzen muss."
- Difficulty level of this description: 2
- Further information needed: no
- Time taken for this description: 1 minutes, 24 seconds.

Description of preconditions, ramifications and goal conditions

- The results of this description can be seen in table A.9.
- A BT, expanded from the given action set can be seen in figure A.5

- The definition of “Close” had to be clarified.
- The precondition *BallClose* has been given for the action *FindBall* but erased after the clarification of “Close”.
- The meaning of precondition and ramification had to be explained: “Diese Begrifflichkeiten sind mir ein bisschen unterschiedlich.”
- The condition *BallFound* was initially not given as precondition for the action *ApproachBall*, but added after clarifying the difference of precondition and ramification.
- Furthermore, it had to be explained that the actions have to be defined separately. *BallFound* as precondition for *ApproachBall* was seen as already given, because of it being a ramification for the action *FindBall*.
- The condition *BallGrasped* was initially given as precondition for the action *ApproachBin* but erased after realizing that it was not necessary for this action: “Obwohl, zum Eimer gehen, das hat ja nichts mit dem Ball zu tun.”
- The participant was not accustomed to this kind of definition: “Das ist eine ganz andere Logik.”
- “Mal kucken, jetzt denke ich bestimmt wieder falsch.”
- “Aber ich musste erst mich losmachen von dem, dass der da hingeht. Da hab ich so ein Bild: der rollt dann da hin, der holt das dann. (...) Erst findet er den Ball, dann geht er zum Ball, dann greift er ihn, aber das sind ja alles Einzelaktionen. Da musste ich mich erst von losmachen. Das war schwierig.”
- When asked about the difficulty level for this description: “5, weil ich eine ganz andere Sache im Kopf hatte. Eine andere Logik.”
- Difficulty level of this description: 5
- Acceptable number of questions for this task: 7
- Time taken for this description: 10 minutes, 57 seconds.

	Preconditions	**Ramifications**
Find Ball		BallFound
Approach Ball	BallFound	BallClose
Grasp Ball	BallFound, BallClose	BallGrasped
Approach Bin		BinClose
Place Ball	BallFound, BallGrasped, BallClose, BinClose	BallPlaced

Table A.9.: Table of preconditions and ramifications filled in by participant 8.

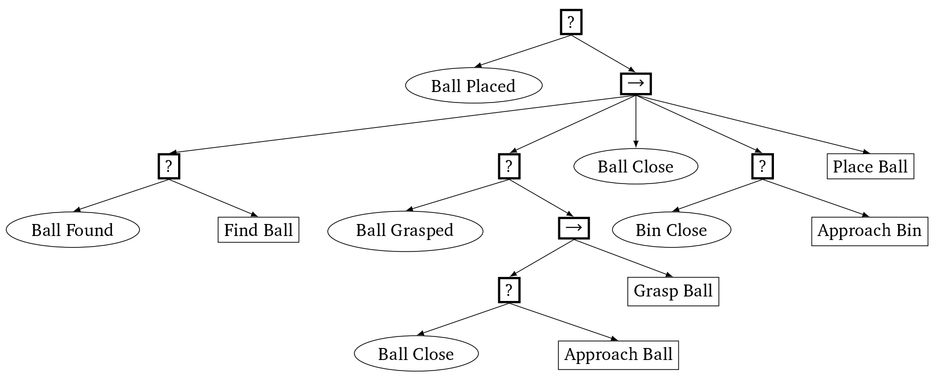


Figure A.5.: BT generated from the action set given by participant 8, seen in table A.9 with approach 1: PA-BT with given actions and goal condition *BallPlaced*.