Symbolic Task Compression in Structured Task Learning

Matteo Saveriano
Institute of Robotics
and Mechatronics
German Aerospace
Center (DLR)
Weßling, Germany
matteo.saveriano@dlr.de

Michael Seegerer
Human-Centered
Assistive Robotics
Technical University
of Munich (TUM)
Munich, Germany
michael.seegerer@tum.de

Riccardo Caccavale
and Alberto Finzi
Dipartimento di Ingegneria
Elettrica e Tecnologie
dell'Informazione
Università di Napoli
Federico II
Naples, Italy
{name.surname}@unina.it

Dongheui Lee
Human-Centered
Assistive Robotics
Technical University
of Munich (TUM) and
Institute of Robotics
and Mechatronics
German Aerospace Center (DLR)
dhlee@tum.de

***Abstract*—Learning everyday tasks from human demonstrations requires unsupervised segmentation of seamless demonstrations, which may result in highly fragmented and widely spread symbolic representations. Since the time needed to plan the task depends on the amount of possible behaviors, it is preferable to keep the number of behaviors as low as possible. In this work, we present an approach to simplify the symbolic representation of a learned task which leads to a reduction of the number of possible behaviors. The simplification is achieved by merging sequential behaviors, i.e. behaviors which are logically sequential and act on the same object. Assuming that the task at hand is encoded in a rooted tree, the approach traverses the tree searching for sequential nodes (behaviors) to merge. Using simple rules to assign pre- and post-conditions to each node, our approach significantly reduces the number of nodes, while keeping unaltered the task flexibility and avoiding perceptual aliasing. Experiments on automatically generated and learned tasks show a significant reduction of the planning time.**

I. Introduction

Human activities can be hierarchically decomposed into a set of behaviors starting from an abstract behavior, like prepare a certain receipt, and adding incrementally more specific behaviors until the atomic actions constituting the task are reached. We refer to such hierarchical tasks as structured tasks. In robotics and AI, structured tasks are often symbolically represented using graph or tree structures with logical pre- and post-conditions associated to each node. The symbolic representation is exploited to plan the robotic task execution.

In order to simplify robot programming, researchers focused on learning tasks from human demonstrations and contextual information [1]–[9]. These approaches are effective in learning symbolic and robot-independent task representations from observation. Learned task structures allow a consistent task execution, are able to generalize the task execution to different contexts [8]–[10], and can be incrementally updated [4], [7].

A drawback of the aforementioned approaches for task structure learning is that they rely on a set of pre-programmed primitive movements for robot execution. The framework presented in our previous works [11], [12], instead, enables simultaneous segmentation and labeling of the human demonstration exploiting supervisory attention mechanisms [13]–[16] to relate the labeled actions to a partially specified task structure. Concurrently, segmented trajectories are encoded into dynamical systems used to generate robot commands.

Regardless the approach used to learn a symbolic representation of the robotic task, the time needed to execute the task depends on the complexity of the associated hierarchical structure, which is executed by a suitable behavior tree. In this setting, the number of nodes/behaviors may easily grow with the number of activities affecting the system performance. For this purpose, this paper extends the framework by [11], [12] to reduce the number of nodes in a learned structure, while preserving a consistent execution. Our method explores the task structure searching for sequential behaviors to merge, while keeping task execution consistency. We show that our approach improves the performance of the framework by [11], [12] in terms of execution time and memory requirements.

II. Structured Tasks Learning and Execution

In this section, we describe the framework introduced in [11], [12] to learn and execute structured tasks via human demonstration and interaction. The system integrates an Attentional System, which monitors and orchestrates high-level tasks, with a lower-level system (Robot Manager), which is responsible of motion learning and execution. These modules are better detailed below.

A. Robot Manager

The Robot Manager (RM) interfaces directly with the robot controller and it is responsible for low-level aspects of the task learning process. The RM smoothly switches between gravity compensation and Cartesian impedance control to allow kinesthetic teaching [17] during the demonstration phase and an accurate task execution. During the human teaching, the RM is capable of on-line segmenting the robot trajectories into basic point-to-point motions assigning them a unique label. The learned motions primitives, represented as stable

dynamical systems, are then used to generate motor commands in the execution phase. The robot manager also performs workspace monitoring activities, like objects classification and tracking, and robot–object distance calculation.

B. Attentional System

Inspired by the way humans orchestrate their own activities [14], [15], the attentional system (AS) exploits two mechanisms to supervise robot actions during both teaching and execution: *i)* contention scheduling manages the execution of routinized activities and allows fast response to external changes, and *ii)* the supervisory attentional system, which regulates the execution of novel responses along with complex and goal oriented behaviors. These two mechanisms are managed, respectively, by the Attentional Executive System and the Attentional Behavior-based System.

The Long Term Memory (LTM) of the Attentional Executive System stores hierarchical descriptions of tasks. These are represented by predicates of the form $\mathbf{schema}(m, l, p)$, where m is the name of the task, l is a list of m_i subtasks associated with enabling preconditions r_i (releasers), i.e. $l{=}\langle(m_1, r_1), \ldots, (m_n, r_n)\rangle$, while p is a post-condition used to check the accomplishment of the task. These definitions are used to allocate and instantiate tasks in the system Working Memory (WM) in order to be executed. All the known tasks are stored in the LTM and, at run-time, tasks to be executed are loaded and maintained in the WM. These running tasks are represented in the WM as rooted trees (see Fig. 1). Each node in the tree is a *behavior*, i.e. a running process associated with an activation value, as well as logical pre- and post-conditions (green and blue ovals in Fig. 1). A customized behavior, the so-called *alive* [13], periodically checks for active behaviors (behaviors with *true* preconditions). In general, multiple behaviors can be active at the same time, hence they compete for the execution. In order to solve this ambiguity, the WM exploits the activation value of each behavior e_b to select the most active behavior using a winner-take-all strategy. As in [14], in our framework each node in the WM has an activation value, which is regulated by top-down and bottom-up mechanisms. The bottom-up regulation $\lambda_b \in [0, 1]$ is a function of behavior-specific stimuli and behavioral state. In line with previous work [11]–[13], in this paper we assume λ_b proportional to the robot–object distance, but more complex regulations are possible [18]. The top-down regulation is $\mu_b = \mu_f + n$, where μ_f is the activation of the parent node and n is the number of accomplished sub-behaviors. The top-down regulation facilitates the selection of active behaviors representing the continuation of accomplished subtasks. The overall activation value (emphasis) is computed as $e_b = \mu_b/\lambda_b$.

C. Motion Primitives Segmentation and Learning

Seamless demonstrations are segmented considering robot–object distance and human commands (open/close the gripper). As shown in Fig. 1, segmentation is triggered if the robot's end-effector enters or leaves the proximity region (a sphere of radius $0.1\,\mathrm{m}$) of an object, or if the teacher commands

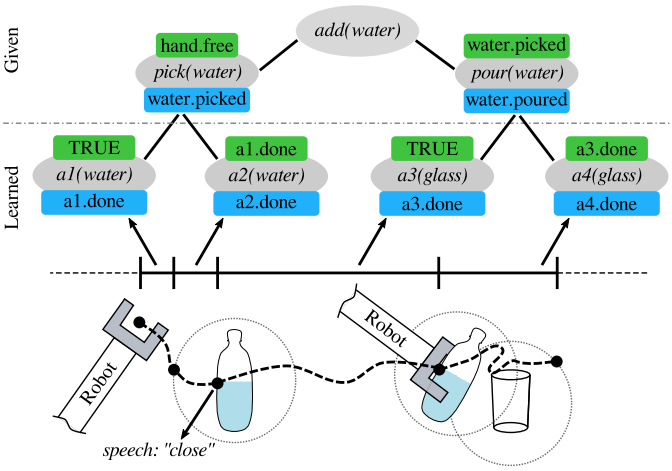

Figure 1. The process used to learn the structured task of pouring water in a cup. (Bottom) The task demonstration is segmented into basic motions. (Top) Symbolic actions are connected to a partially specified task structure.

to open/close the gripper. Segmented pose trajectories are compactly represented as stable dynamical systems, the so-called Dynamic Movement Primitives (DMP) [19], which are used to generate the on-line motion. Dynamical systems are effective in motion generation due to their convergence properties and to their robustness to external perturbations like unforeseen obstacles [20]–[22]. During the demonstration, the RM assigns a unique label to each segment (the symbolic actions *a1* to *a4* in Fig. 1), calculates the closest object, and communicates the attentional system that a new symbolic action has been performed on a certain (the closest) object.

D. Task Learning and Execution

The attentional system (AS) connects the generated symbolic actions with a partially specified task structure. As shown in Fig. 1, the abstract structure of the demonstrated task is given and allocated in the WM. During the teaching phase, the AS assigns the segmented symbolic action to the enabled subtask (true pre-condition) with the higher activation value. Moreover, the precondition of the new action is set to *true* (action always enabled) if the subtask node has no children. If at least one action is already attached to the subtask, the new action is enabled after the execution of the previous one. By defining the pre-condition in this way, actions are ordered as they are demonstrated, but other choices are possible. As for the execution phase, the AS exploits pre-conditions and emphasis to determine the most emphasized action and to command its execution. The AS periodically checks for the most emphasized action, which allows the instantaneous adaptation of the execution sequence to the operative context.

III. Symbolic Task Compression

A. Problem Description

The learning procedure described in. Sec. II-C and II-D is effective for learning and reproducing structured tasks via imitation learning. In Fig. 2 (left) we show an example of

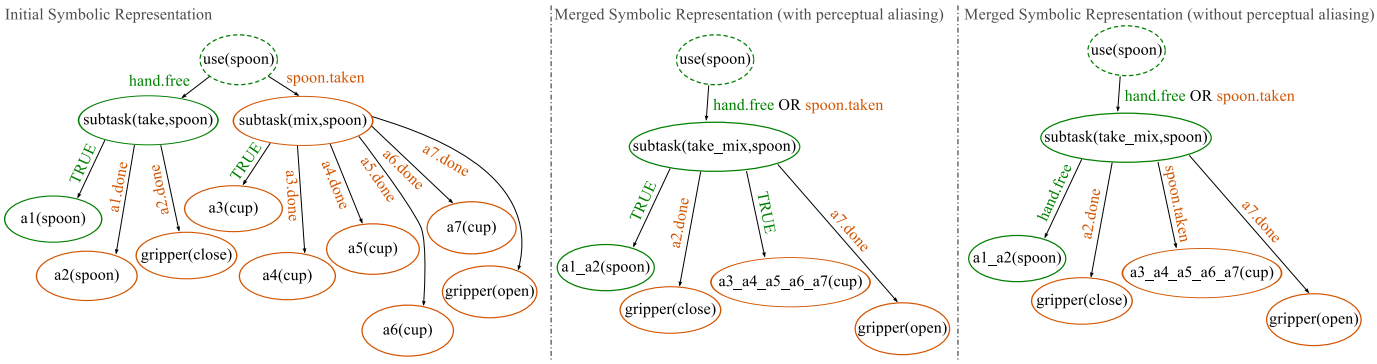

Figure 2. The structure simplification approach applied to the task of mix the content of a cup with a spoon. Green (brown) text indicates a *true* (*false*) pre-condition, while a green (brown) ellipse indicates an active (inactive) behavior. (Left) The symbolic representation learned using the approach by [12]. (Middle) The merged symbolic representation with *perceptual aliasing* between *a1_a2(spoon)* and *a3_a4_a5_a6_a7(cup)* (both actions have a *true* pre-condition). (Right) The propagation of the father's (subtask) pre-condition to the child nodes (actions) prevents the perceptual aliasing.

learned task where the robot has to take the spoon and mix the liquid into the cup. In this case, the proposed segmentation mechanism produces three segments for the take-spoon task: the robot has to reach the proximity region of the spoon firstly (*a1(spoon)*, equivalent to a pre-grasping pose), then the grasping pose (*a2(spoon)*), and to close the gripper. Although this structure, along with the associated pre- and post-conditions, allows to execute the demonstrated trajectories in the right order[1], it produces a large number of nodes that can be suitably compressed without altering the task execution. This problem is of particular interest in this context, because the attentional system periodically checks the task execution state to command the most emphasized action, and the time needed to compute the most emphasized action grows with the number of nodes in the tree. It is worth noting that the learning framework introduced in Sec. II exploits an already provided partially specified task structure and connects the segmented actions to this structure during the teaching. Hence, compression may be limited to the low-level segmented activities (leaves of the tree). Nevertheless, the compression algorithm proposed here can merge behaviors at all levels of the tree, because abstract specification of tasks may be provided by another learning process [2], [8] or encoded in order to be human readable, but not optimized for robot execution.

B. Merging Sequential Behaviours

Our merging approach aims at preserving task consistency, while reducing the number of nodes in the tree. Specifically, we merge two behavior nodes if they are *sequential* according to the definition introduced below.

Definition 1: Two behaviors b_1 and b_2 of the WM tree are *sequential* if the following conditions are contemporary met:

1) b_1 and b_2 are children of the same node.
2) b_1 and b_2 are performed by the same agent.
3) b_1 and b_2 act on the same objects (i.e. arguments representing target objects are equally instantiated).
4) The post-condition of b_1 equals the pre-condition of b_2 or the post-condition of b_2 equals the pre-condition of b_1.

[1]Note that the post-condition of an action *ai* is *ai-1.done*, the post-condition of *gripper(close)* is *not hand.free*, the post-condition of *subtask(take,obj)* is *obj.taken*. We omit the post-conditions in Fig. 2 for a better visualization.

Algorithm 1 Simplify the symbolic representation

1: $beh\_list \leftarrow$ loadTaskFromLTM(task_label)
2: $new\_beh\_list \leftarrow \{\}$ // Empty behavior list
3: **while** beh_list is not empty **do**
4: $b_i \leftarrow beh\_list.pop()$ // Get first behavior in the list
5: $seq\_beh\_list \leftarrow$ findSequentialBehaviors(beh_list, b_i)
6: **for** b_j in seq_beh_list **do**
7: $b_m \leftarrow genNewNode$ // New behavior node
// Merge sequential behaviors
8: $b_m.label \leftarrow$ createUniqueLabel(b_i, b_j)
9: $b_m.child \leftarrow \{b_i.child, b_j.child\}$
10: $b_m.pre\_cond \leftarrow b_i.pre\_cond\ OR\ b_j.pre\_cond$
11: $b_m.post\_cond \leftarrow b_j.post\_cond$
// Avoid perceptual aliasing
12: **for** c_i in b_i.child **do**
13: $c_i.pre\_cond \leftarrow c_i.pre\ AND\ b_i.pre\_cond$
14: **end for**
15: **for** c_j in b_j.child **do**
16: $c_j.pre\_cond \leftarrow c_j.pre\ AND\ b_j.pre\_cond$
17: **end for**
// Update task structure
18: $new\_beh\_list.push(b_m)$
19: $beh\_list.push(b_m)$ // Store new behavior to check for chains
20: **end for**
21: **end while**

The task compression method is summarized in Alg. 1. The first step of our procedure is to instantiate a task given its specification in LTM and collect all the nodes of the hierarchical structure starting from the root node. Each node of the tree is associated with a name (unique label) and a list of directly associated sub-behaviors (child nodes) (see Sec. II-B). The result of $loadTaskFrom(Task_label)$ (line 1 in Alg. 1) is a list of behaviors candidates for merging.

Starting from the root node, in the second step we explore one by one the behaviors in beh_list looking for sequential

behaviors (lines 3–5 in Alg. 1). The simplest case is when only two sequential behaviors b_i and b_j are encountered, i.e. the function "findSequentialBehaviors" in Alg. 1 returns a list with one element b_j. The two sequential behaviors are then merged into one behavior b_m and a unique label is assigned to the merged behavior (e.g. *subtask(take_mix,spoon)* in Fig. 2 (middle)). In general, a chain of more than two sequential behaviors may exist, like the actions *a3* to *a7* in Fig. 2 (left) where *a4* has the post-condition of *a3* as pre-condition, *a5* has the post-condition of *a4* as pre-condition and so on. In this case, the merging is iteratively repeated until all the sequential behaviors are merged into one—action *a3_a4_a5_a6_a7(cup)* Fig. 2 (middle); that is, if we assume that b_i and b_j, as well as b_j and b_k, are sequential behaviors, the algorithm will first generate the merged behaviors $b_i\_b_j$ and $b_j\_b_k$, which are still sequential and are merged into one behavior $b_i\_b_j\_b_k$. Another possibility is that multiple behaviors share the same pre-condition, for example b_j and b_k have the post-condition of b_i as a pre-condition, in this case, the algorithm produces two merged behaviors, e.g. $b_i\_b_j$ and $b_i\_b_k$. In general, if $N-1$ behaviors have the post-condition $b_i.post\_cond$ as pre-condition, the algorithm generates the $N-1$ merged behaviors $b_i\_b_2, \ldots, b_i\_b_N$. The described merging steps corresponds to lines 6–11 in Alg. 1, while lines 18–19 enables the iterative merging process.

In all the possible cases, the described procedure (repeatedly) merges two sequential behaviors. Once a merging occurs, the generated sequence behavior (line 15) inherits the child nodes of the two original behaviors; therefore, the children lists of each behavior $b_i.child$ and $b_j.child$ are also merged into a unique list and stored in $b_m.child$ (line 9). Note that, in the example in Fig. 2, we are not considering *gripper(close)* and *gripper(open)* sequential to other motion actions. This is because we consider the robot arm and the gripper different agents that perform different kind of actions, i.e. motion actions for the arm and grasp actions for the gripper.

The next problem to consider is the allocation of pre- and post-conditions to the new behavior. The new pre- and post-conditions has to preserve both the flexibility of the symbolic representation and the execution constraints. For instance, looking at the mix with a spoon task in Fig. 2 (left), we can merge the sequential behaviors *subtask(take,spoon)* and *subtask(mix,spoon)*, but we do not want to lose the possibility of performing only the mix action if the spoon is already in the robot gripper. To this end, the pre-condition of a new behavior is a combination of the pre-conditions of the merged behaviors linked with a logical *OR* operator (line 10). As shown in Fig. 2 (middle), the behaviors *subtask(take,spoon)* and *subtask(mix,spoon)* are merged into *subtask(take_mix,spoon)* which is activated if one of *hand.free* and *spoon.taken* is *true*. It is clear that if one of the pre-conditions is always *true*, as for *a1(spoon)*, the *OR* can be omitted and the pre-condition of the merged behavior is directly *true*. Finally, the post-condition of the new generated task is set to the post-condition of the last merged task (line 11). As an example, the new behavior *a1_a2(spoon)* in Fig. 2 (middle) has *a2.done* as post-condition which enables the execution of *gripper(close)*.

C. Avoid Perceptual Aliasing

In our setting, perceptual aliasing occurs when multiple behaviors are enabled in the same state and no knowledge is available to enable the correct execution of a task [23], [24]. In the approach described in Sec. II, perceptual aliasing is not an issue. Even if multiple behaviors can be active at the same time, the winner-take-all approach allows the selection of the most active behavior, in so resolving the impasse. On the other hand, the pre-condition of the merged behavior (line 10) enables behaviors execution in the wrong operative context. For instance, in Fig. 2 (middle), *a1_a2(spoon)* and *a3_a4_a5_a6_a7(cup)* are always active due to the *true* pre-condition. In this case, the system will exploit the activation values to select the most active among them, without considering if the spoon has been previously taken or not; here, the execution of *a3_a4_a5_a6_a7(cup)*, which corresponds to mix in the cup and place the spoon, before grasping the spoon causes a failure. However, in our framework, this issue can be avoided by propagating the pre-condition of the father node to the children before merging. The pre-condition of the child node(s) inherits also the pre-condition of the father node (lines 12–17 in Alg. 1) and the merging continues as described in Sec. III-B. The results of this procedure are illustrated in Fig. 2 (right). The actions *a1_a2(spoon)* and *a3_a4_a5_a6_a7(cup)* have now *hand.free* and *spoon.taken* as pre-conditions respectively and the attentional system will properly sequence the execution of the task.

D. Top-down and Bottom-up Regulations

In the proposed approach, behavior merging also affects the propagation of the system activation values. We assume that bottom-up regulations are proportional to the robot-object distance, therefore the merged behaviors share the same regulation, i.e. $\lambda_m = \lambda_i = \lambda_j$, with b_m merges b_i and b_j. For top-down regulations, the regulation mechanism is slightly changed, i.e. $\mu_{b_m} = \mu_f + m$, where μ_f is the activation value of the parent node, m represents the number of accomplished sub-behaviors in the original tree. In order to update μ_f, in the compressed tree, each behavior is associated with a value k that represents the number of merged behaviors from the original tree; when a sub-behavior is accomplished in the compressed tree, μ_f is incremented by k.

IV. Evaluation

A. Reduction Degree

The goal of the evaluation is to experimentally verify that the proposed approach reduces the time needed to execute the next action, which consists of traversing the tree to find active leaves and selecting the most emphasized action (see Sec. II-B) It is clear that the execution time depends on the number of behaviors (nodes of the tree)[2], hence the performance of the simplification algorithm depends on the number of nodes

[2]In our implementation each node is instantiated as a separate thread and periodically executed.

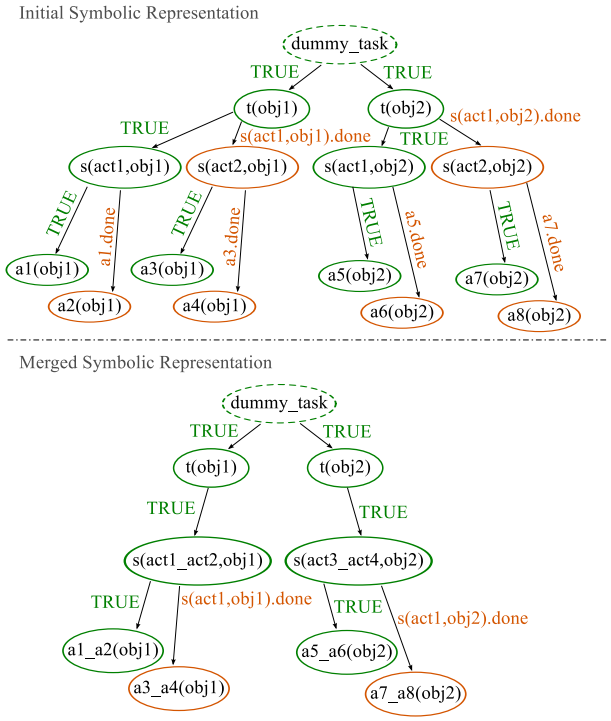

Figure 3. (Top) The task representation obtained with a depth of 4, 2 children per node, and a desired $RD = 0.5$. (Bottom) The merged task tree.

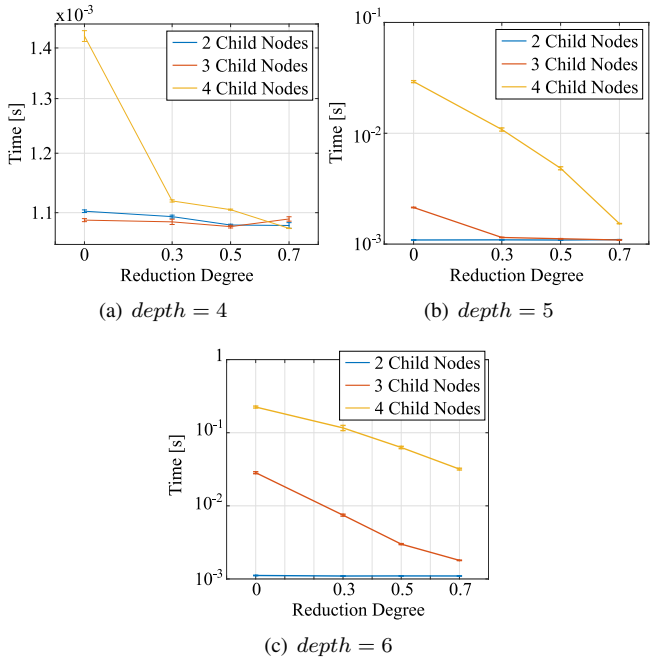

Figure 4. The execution time as a function of the reduction degree obtained for automatically generated trees with different depth ($D = 4, 5, 6$) and children per node (2, 3, or 4).

in the merged tree. In order to quantitatively measure the performance, we introduce the *Reduction Degree* (RD) which indicates the percentage of sequential nodes in a tree, i.e. the percentage of nodes that are going to be merged. The *reduction degree* is computed as

$$RD = (B_i - B_m)/(B_i - D), \tag{1}$$

where $B_i \in \mathbb{N}$ is the number of nodes (behaviors) in a tree before merging, $B_m \in \mathbb{N}$ is the number of nodes after applying Alg. 1, and $D \in \mathbb{N}$ is the depth of the tree. Note that, in general, a tree with depth D has at least one node for each level ($B_i = D$). In our case, it is reasonable to assume that $B_i > D$, which means that, for some levels, there is more than one possible behavior. If $B_i > D$ the reduction degree in (1) is positive definite ($B_i \geq B_m$). It is zero only if $B_i = B_m$ (no behaviors are merged), and it saturates to one for $B_m = D$.

B. Artificially Created Tasks

In this setting, we randomly create tree structures with a desired reduction degree. To automatically construct the trees, we consider a depth of $D = 4, 5, 6$, which are reasonable values to represent typical human tasks [12] assuming that each node in the tree has 2, 3, or 4 children. Considering these combinations of depths and child nodes, we obtain 9 tree structures with a different amount of behaviors (from 15 to 1365). As shown in Fig. 3 (top), a tree with $D = 4$ and 2 children has 15 nodes: 1 root node, 2 nodes at the second level, 4 nodes (2 for each father) at the third level, and 8 leaves. Given a tree structure and a desired reduction degree, we assign pre- and post-conditions to each node in order to have a desired number of sequential nodes. Specifically, given a tree with B_i nodes and depth D, assuming a reduction degree RD, we can invert (1) to get the number of nodes in the merged structure B_m. Once we have the number of nodes to merge ($B_i \rightarrow B_m$) we can introduce the associated sequential nodes by properly assigning pre- and post-conditions. More specifically, the root has always one node. The second level from the top has 2 to 4 nodes, depending on the desired number of children. We assign a *true* pre-condition to these nodes, i.e. we decided not to merge the second level since it contains few nodes. In order to solve the conflicts generated by the *true* pre-conditions, each node/behavior of the second level acts on a different object ($obj_i, i = 1, \ldots, 4$). From the third level on, each node b_i has two possible pre-conditions, namely *true* (always active) or $b_j.done$ (active if another behavior b_j is executed), and one post-condition $b_i.done$. The two pre-conditions are assigned to create approximately $2(B_i - B_m)$ sequential nodes and match the desired RD. Also in this case, always active behaviors are assumed to act on different objects to resolve the conflicts. For each tree, we performed 10 executions in order to evaluate the performance with and without the compression.

The times observed with different depths ($D = 4, 5, 6$), children per node (2, 3, and 4), and reduction degrees ($RD = 0, 0.3, 0.5, 0.7$) are shown in Fig. 4. As expected, the execution time reduces with the reduction degree. As shown in Fig. 4(a), the merging approach has no significant benefits up to 40 nodes. With 85 nodes and a reduction degree of 0.3, we observe a reduction of the execution time of the 20% (from

1.4 ms to 1.1 ms). With the maximum number of nodes in the considered scenario ($B_i = 1365$), the execution time drops from 0.23 s with $RD = 0$ to 0.03 s with $RD = 0.7$ (Fig. 4(c)) without loosing any relevant information on the task constraints. The highest reduction of the execution time is the 93% observed in Fig. 4(b) (yellow line) and Fig. 4(c) (brown line). Finally, results show that the influence of the depth to the execution time is minimal.

C. Learned Tasks: Prepare Tea and Coffee

We now evaluate the effectiveness of our approach on a real task of preparing a cup of tea and a cup of coffee. The task is obtained by combining the separate tasks *prepare_cofee* and *prepare_tea* learned by demonstration as in [12]. The root of the tree is then the joined task *prepare_coffe_and_tea*. Before merging, the task structure has a depth of 5 and a total of 63 nodes. After merging, the task structure has the same depth of 5 and a total of 33 nodes. Hence, the reduction degree of the structure, calculated using (1), is about $RD \approx 0.5$. By merging sequential nodes, the execution time reduces hereby from 1.5 ms to 1.1 ms (a reduction of 27%). In order to simulate the task execution, we emulate the robot manager by acknowledging the execution of the last commanded action, and letting the attentional system plan the next action. The task is executed 10 times with different object configurations. In all the performed tests the task is successfully accomplished. Hence, although the correctness of the merging algorithm has not been theoretically proved, we experimentally find that the represented task is correctly executed also with the merged tree. Obtained results suggest that the proposed method is effective and its efficacy increases with the amount of nodes which can be merged.

V. Conclusion and Future Work

In learning by demonstration, unsupervised activity segmentation can produce fragmented representation of activities and tasks, which can affect the performance of overall system. In this work, we tackled this problem proposing an approach that provides compact representations of the learned tasks at different levels of abstraction. The presented method exploits the precondition and effect constraints associated with the tasks/activities in order to detect and merge sequential behaviors while preserving the flexibility of the original structure and avoid perceptual aliasing. The conducted evaluation shows that our approach significantly reduces the task planning time, especially with large task structures.

A possible drawback of the proposed approach is that a compressed tree cannot always be decompressed, which may lead to failures if the correct execution of each merged action is important to control the task execution. In other words, if two actions are merged into one, the system cannot check for the correct execution of both the actions and, in case of failure, replan only the failed action. Moreover, in order to execute merged tasks on a real robot, the originally learned motion primitives have to be merged as well. Solving those issues will be the topic of our future research.

Acknowledgment

This work has been supported by Helmholtz Association.

References

[1] S. Calinon and D. Lee, "Learning control," in *Humanoid Robotics: a Reference*, P. Vadakkepat and A. Goswami, Eds. Springer, 2018.

[2] R. Dillmann, "Teaching and learning of robot tasks via observation of human performance," *Rob. Auton. Syst.*, vol. 47, no. 2, pp. 109–116, 2004.

[3] G. Konidaris, S. Kuindersma, R. Grupen, and A. Barto, "Robot learning from demonstration by constructing skill trees," *IJRR*, vol. 31, no. 3, pp. 360–375, Mar. 2012.

[4] M. Pardowitz, S. Knoop, R. Dillmann, and R. D. Zöllner, "Incremental learning of tasks from user demonstrations, past experiences, and vocal comments," *Trans. Syst., Man, Cybern. B, Cybern.*, vol. 37, pp. 322–332, 2007.

[5] Y. Kuniyoshi, M. Inaba, and H. Inoue, "Learning by watching: extracting reusable task knowledge from visual observation of human performance," *Trans. Robot. Autom.*, vol. 10, no. 6, pp. 799–822, 1994.

[6] R. Cubek, W. Ertel, and G. Palm, "High-level learning from demonstration with conceptual spaces and subspace clustering," in *ICRA*, 2015, pp. 2592–2597.

[7] I. Dianov, K. Ramìrez-Amaro, P. Lanillos, E. Dean-Leon, F. Bergner, and G. Cheng, "Extracting general task structures to accelerate the learning of new tasks," in *Humanoids*, 2016, pp. 802–807.

[8] M. N. Nicolescu and M. J. Mataric, "Natural methods for robot task learning: Instructive demonstrations, generalization and practice," in *AAMAS*, 2003, pp. 241–248.

[9] T. Abbas and B. A. MacDonald, "Generalizing topological task graphs from multiple symbolic demonstrations in programming by demonstration (pbd) processes," in *ICRA*, 2011, pp. 3816–3821.

[10] G. Gemignani, S. D. Klee, M. Veloso, and D. Nardi, "On task recognition and generalization in long-term robot teaching," in *AAMAS*, 2015, pp. 1879–1880.

[11] R. Caccavale, M. Saveriano, G. A. Fontanelli, F. Ficuciello, D. Lee, and A. Finzi, "Imitation learning and attentional supervision of dual-arm structured tasks," in *ICDL-EPIROB*, 2017, pp. 66–71.

[12] R. Caccavale, M. Saveriano, A. Finzi, and D. Lee, "Kinesthetic teaching and attentional supervision of structured tasks in human–robot interaction," *Autonomous Robots*, 2018.

[13] R. Caccavale and A. Finzi, "Flexible task execution and attentional regulations in human-robot interaction," *TCDS*, vol. 9, no. 1, pp. 68–79, 2017.

[14] D. A. Norman and T. Shallice, "Attention to action: Willed and automatic control of behavior," in *Consciousness and self-regulation: Advances in research and theory*, 1986, vol. 4, pp. 1–18.

[15] R. P. Cooper and T. Shallice, "Hierarchical schemas and goals in the control of sequential behavior," *Psychological Review*, vol. 113, no. 4, pp. 887–916, 2006.

[16] R. Caccavale and A. Finzi, "Plan execution and attentional regulations for flexible human-robot interaction," in *SMC*, 2015, pp. 2453–2458.

[17] D. Lee and C. Ott, "Incremental kinesthetic teaching of motion primitives using the motion refinement tube," *Autonomous Robots*, vol. 31, no. 2, pp. 115–131, 2011.

[18] X. Broquère, A. Finzi, J. Mainprice, S. Rossi, D. Sidobre, and M. Staffa, "An attentional approach to human–robot interactive manipulation," *Int. J. Soc. Robot.*, vol. 6, no. 4, pp. 533–553, 2014.

[19] D.-H. Park, H. Hoffmann, P. Pastor, and S. Schaal, "Movement reproduction and obstacle avoidance with dynamic movement primitives and potential fields," in *Humanoids*, 2008, pp. 91–98.

[20] M. Saveriano and D. Lee, "Point cloud based dynamical system modulation for reactive avoidance of convex and concave obstacles," in *IROS*, 2013, pp. 5380–5387.

[21] ——, "Distance based dynamical system modulation for reactive avoidance of moving obstacles," in *ICRA*, 2014, pp. 5618–5623.

[22] M. Saveriano, F. Hirt, and D. Lee, "Human-aware motion reshaping using dynamical systems," *Pattern Recognition Letters*, vol. 99, pp. 96–104, 2017.

[23] S. Manschitz, J. Kober, M. Gienger, and J. Peters, "Learning movement primitive attractor goals and sequential skills from kinesthetic demonstrations," *Rob. Auton. Syst.*, vol. 74, pp. 97–107, 2015.

[24] S. D. Whitehead and D. H. Ballard, "Learning to perceive and act by trial and error," *Machine Learning*, vol. 7, no. 1, pp. 45–83, 1991.