BULGARIAN ACADEMY OF SCIENCES

CYBERNETICS AND INFORMATION TECHNOLOGIES • Volume **12**, No 3

Sofia • 2012

# Learning to Generalize from Demonstrations

*Katie Browne, Monica Nicolescu*

*University of Nevada, Reno, NV 89557, USA*
*Emails: katiembrowne@gmail.com monica@cse.unr.edu*

***Abstract:*** *Learning by demonstration is a natural approach that can be used to build a robot's task repertoire. In this paper we propose an algorithm that enables a learner to generalize a task representation from a small number of demonstrations of the same task. The algorithm can generalize a wide range of situations that typically occur in daily tasks. The paper also describes the supporting representation that we use in order to encode the generalized representation. The approach is validated with experimental results on a broad range of generalizations.*

***Keywords:*** *Learning by Demonstration, Generalized Representation, Graph Task Representation, Behavior Graphs, Robotics.*

## 1. Introduction

The ability to learn new knowledge is essential for any robot to be successful in real-world applications where it would be impractical for a robot designer to endow it with all the necessary task capabilities that it would need during its operational lifetime. Therefore, it becomes necessary that the robot is able to acquire this information in a human-like teaching approach. While significant research has been performed in the area of learning motor behavior primitives from a teacher's demonstration, the topic of learning of the general task knowledge has not been sufficiently addressed. In learning general task knowledge, the main challenge for the learner is to extract all the necessary information pertaining to the task, eliminate all the observations that are irrelevant and generalize the correct task

representation in the case when multiple, possibly different demonstrations are given. In this paper we propose an algorithm that enables generalization of correct and complete task knowledge from a small number of demonstrations of the same task, under the assumption of possibly very different demonstrations provided to the learner.

In our proposed approach we will assume that a learner robot is equipped with a basic set of capabilities or skills; the robot is aware of the goals each of these skills accomplish and also under which conditions these skills can or should be executed. During the learning stage, the learner robot is presented with a set of training examples with which it makes a generalized representation graph structure for each of the skills. These generalized representation graphs are then used in the execution part to create an execution sequence that achieves the task at hand.

A significant advantage of the proposed method is these generalized representation graphs are always consistent with the training samples. In addition, our approach allows the robot to execute the learned tasks in collaboration with a human user: the user can help the robot by performing some parts of the task and the robot takes into account the user's actions in order to finish up the task.

The remainder of the paper is structured as follows: Section 2 summarizes related work in the field of learning by demonstration. Section 3 presents the generalization problems this paper is attempting to address and it describes the proposed method used to solve these problems. Section 4 presents the experimental results, Section 5 discusses these results and future work, and finally Section 6 concludes our paper.

## 2. Related work

A significant challenge in designing robot systems that learn from demonstration is the interpretation of observations gathered from the instruction, as the robot has to process the continuous stream of data coming from its sensors, and then translate it into appropriate skills or tasks. In most cases this consists of segmenting the data stream into meaningful units, and then mapping them to a set of existing behavior primitives (Schaal [1]). There is a large spectrum of approaches to the problem of segmentation, including Principal Component Analysis (PCA) (Voyles, Morrow & Khosla [2]), gesture interpretation (Voyles & Khosla [3]), Learning Vector Quantization (LVQ) (Pook & Ballard [4]), motion contact (Tominaga, Takamatsu, Ogawara, Kimura&Ikeuchi [5], Ikeuchi, Kawade & Suehiro [6]) and geometric interpretation of the demonstration (Kuniyoshi&Inoue [7]).

Several recent approaches to learning by demonstration also use a graph/tree representation. Feature decision trees have been used to find the next behavior for a soccer player (Avrahami-Zilberbrand & Kaminka [8]), topological task graphs using longest common sequences has been used in path planning (Abbas&MacDonald [9]), and skill trees have been successfully applied to the pinball domain (Konidaris, Kuindersma, Grupen & Barto [10]).

All of these approaches differ from the generalization graphs proposed in this paper in that the graph/trees represent different elements of the learned tasks.

## 3. Task learning from generalizations

### 3.1. Classes of generalization problems

Approaches to learning by demonstration or from observation are significantly influenced by performance of the demonstrator. The challenge for the learner is to extract all the relevant information pertaining to the task and eliminate all observations that are irrelevant. During learning from demonstrations there are various aspects of the task that could and should be refined through generalizations. In this paper our goal is to address the following generalization problems:

1. *Generalization 1*. Learning to distinguish between:
    a. Tasks for which the ordering of the demonstration steps is important. Under this case, the exact demonstrated path needs to be reproduced.
    b. Tasks for which the ordering of the task's steps is not important, but where achieving the final goal is the main focus of the demonstration. Under this case, multiple ways of achieving the goal could be possible.
    c. Tasks for which the ordering of certain steps is important but the ordering of the rest of the steps is not important. Under this case,
        i. A step or set of steps needs to come before another step or set of steps but is unrelated to the other steps in the task.
        ii. A sequence of steps may need to be performed together but can appear anywhere during the task.
        iii. A step may need to be done at the same point in time.
2. *Generalization 2*. Learning to distinguish which steps in the demonstration are relevant, and which steps do not bring any contribution to the task. This will be done by monitoring the steps that do not appear in every demonstration.
3. *Generalization 3.* Learning to distinguish the number of times a step needs to be repeated since this number may change in each demonstration. This will be done by monitoring the post conditions of the repeated step after all the repetitions.

### 3.2. Collaborative behavior during task execution

Another goal of the proposed approach is to handle situations, in which a teacher or another user helps out the robot that is currently carrying out a learned task. In this case the learner will need to detect that some of the behaviors may have been done while the learner was performing a different behavior in the task. The learner also needs to make sure the behavior that the other person performed met the post conditions of the behavior and if not, the learner will need to complete that behavior.

### 3.3. Representation of generalized tasks

In this work a task is represented as a sequence of symbols where each symbol corresponds to a behavior that a robot can perform. A sequence of such symbols, provided from a teacher's demonstration, constitutes a training sample. In this context, a learner is given a set of sequences, potentially different, that all achieve the same target task. We base our generalization strategy on the preconditions of behaviors shown in the demonstration, based on what other behaviors have occurred prior to their execution. For instance, if behaviors *a* and *b* come before *d* in all training samples, but *c* can appear before or after *d*, we can generalize that behaviors *a* and *b* are preconditions for behavior*d* as they may achieve goals that are necessary for *d*'s execution. Our goal is to build, for each behavior present in the demonstrated task, a generalized representation of behavior preconditions that encapsulates the information contained in all training samples. This representation could afterwards be used by the learner to perform the same observed task.

The representation for generalized preconditions in this approach is a graph that is mainly a tree structure with the added feature that leaf nodes in the graph can contain an edge pointing back to the root of the graph. The root of the graph is a behavior for which we build the generalized preconditions and the branches of the graph are the behaviors that came before the current behavior in the training examples. Thus, each behavior in the training example has its own graph.

To demonstrate how the generalized precondition graphs for a task are created, let us consider an example in which the following training samples have been given for a particular task:

- *abcd*
- *cadb*
- *cdba*

Our approach works by starting at the first behavior in the first sequence which in this case is *a*. A new graph is created with *a* becoming the root node and a node with the name *NULL* added as a child, because no behavior has been executed before *a*, and therefore the behavior has no preconditions. This child node is then marked as final state; the final state signifies that the behavior represented by the root node can be performed after the behavior marked as a final node in the current sequence. Also the number of times a final node appears before the behavior is recorded and is used at the execution stage to decide which behavior to perform in a sequence when multiple behaviors are applicable and can be chosen at the same time. Next the algorithm creates the precondition graph for behavior *b*; the graph starts with *b* as the root node and *a* as a final state child node, because *a* was executed before it in the first training sample. A new graph is then created for behavior *c* and the branch with nodes *a* and *b* is added. Finally the graph for *d* is created and *a*, *b*, and *c* are added as a branch. The graphs created are shown in Fig. 1. The numbers appended to the behaviors in the graph have the following meaning: the first number appended signifies the level that it is located in the graph and the second number is only appended to the final state nodes, representing the number of times the node was a final state in the training samples.

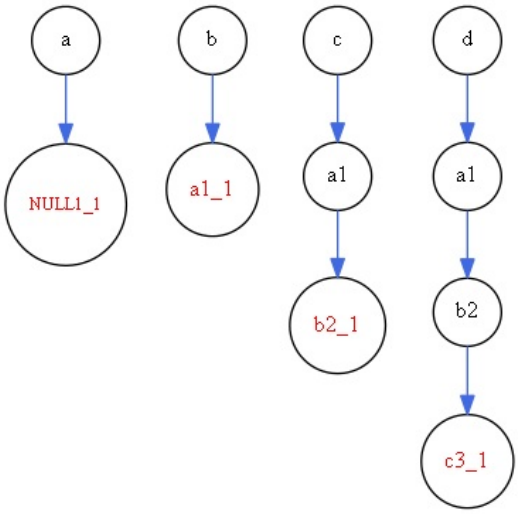


Fig. 1. Graphs created after the first sequence of behaviors

After the second training sequence is added, all the precondition graphs are refined as follows. Behavior*c* is the first in the training sample, so a null child node is added to its graph, indicating that it can also be the first one executed in the task. Then, behavior *c* is added as a child node in behavior *a*'s graph, a branch containing behaviors*c* and *a* are added to *d*'s graph, and finally *c*, *a*, and *d* become a branch in *b*'s graph. Fig. 2 shows the graphs after adding the second training sample.

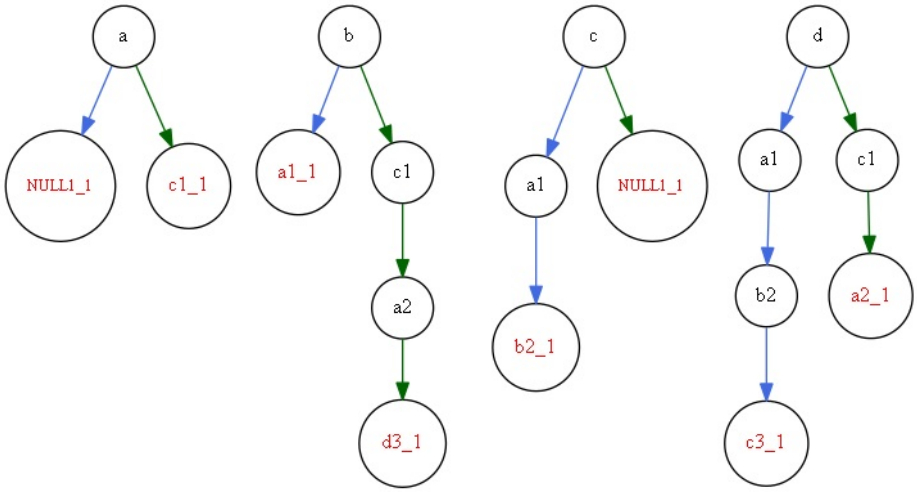


Fig. 2. The behaviour graphs after the second sequence has been added

Lastly the third training sample is incorporated using the same strategy as above. The resulting generalized precondition graphs can be seen in Fig. 3.

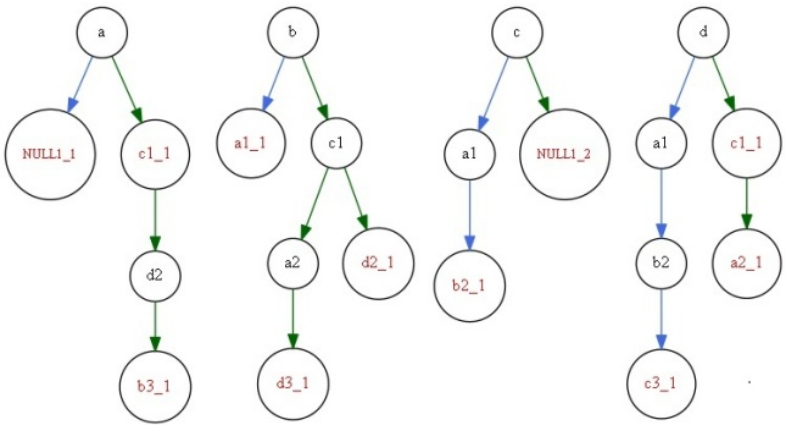

Fig. 3. The behaviour graphs after the third and last sequence has been added

If a behavior occurs in a training sample two or more times, the children nodes in the generalized precondition graph will point back to the parent node. Consider the sequence:

- *abcfdefg*

When a graph for behavior *f* is created, a branch containing the behaviors *a*, *b*, and *c* will be added to *f*'s graph and the *c* node will have an edge pointing to *f*'s node. A branch containing *d* and *e* is also added to the graph. *f*'s graph can be seen in Fig. 4.

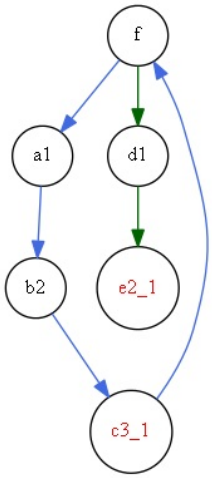

Fig. 4. Example of a graph whose behaviour appears one or more times in a sequence

Additionally, the graphs keep track of the number of training samples each behavior appears in. This number can be used to calculate the percentage of times a behavior appeared in the training example which in turn is used to find behaviors that did not appear in every sample. Depending on this percentage, we can choose to keep the behavior's graph or completely remove it.

Finally, the generalized precondition graphs keep track of whether a behavior appeared two or more times in a training sample. In our representation, these behaviors are aggregated into *special behaviors* which are just behaviors that the

learner repeats in the task as efficiently as possible until the post conditions are met. At the beginning of the learning period, the learner is provided a list of behaviors that can be performed and the possible parameters for these behaviors. Conclusively, these special generalized representation graphs are always consistent with the training samples.

### 3.4. Usage of generalization graphs

The generalized representations we built through the method presented above can be used by the learner to reproduce the task that has been demonstrated. Given that multiple demonstrations have been provided, the learner needs to find an execution sequence that achieves the same goal and is consistent with all the training examples. To do this, the system goes through all the precondition graphs and finds the behaviors that have the current execution sequence as a branch in their graph. If two or more behaviors fit the criteria, we pick the behavior that has been visited the most at the current time behavior in the training examples or if all the behaviors have been visited equally, we randomly pick a behavior as the next one in the sequence. To demonstrate this, considered the previous example and graphs in Fig. 3:

- $t_0$: Current sequence: *null* Behaviors that qualify: *a* and *c* Choose: *c*
- $t_1$: Current sequence: *c* Behaviors that qualify: *a* and *d* Choose: *d*
- $t_2$: Current sequence: *cd* Behaviors that qualify: *b* Choose: *b*
- $t_3$: Current sequence: *cdb* Behaviors that qualify: *a* Choose: *a*
- $t_4$: Current sequence: *cdba*

## 4. Experimental results

The system was tested in the context of a task for making the dough for chocolate chip cookies. A training example was a set of sequences that were made up of strings representative for the behaviors needed for this task. The list of behaviors for this task and the parameters that these behaviors can take are as follows:

Add Sugar (*Su*) 0.5 cup,
Add Salt (*Sa*) 1 tsp,
Add Baking Soda (*BSo*) 1 tsp,
Add Flour (*F*) 0.5, 1, 2 cup,
Add Chocolate Chips (*CC*) 0.5, 1, 2 cup,
Add Brown Sugar (*BSu*) 1, 2 cup,
Add Vanilla (*V*) 1 tsp,
Add Butter (*B*) 0.5, 1, 2 cup,
Add Eggs (*E*) 1, 2 egg,
Add Nuts (*N*) 0.5, 1, 2 cup,
Stir (*St*) 1, 2, 3, 4 time.

The results for our system depended on whether or not the generalized representations produced an execution trace consistent with all the training examples. In most cases, this meant that the order of the behaviors matched one of the sequences given to our system. However, when the examples had repeated

behaviors or behaviors that only appeared in a small percent of the sequences, this was not necessarily the case. When there were repeated behaviors, the number of times the behavior was repeated depended on what amounts the learner could add to meet the post conditions. Furthermore, when a behavior appeared in a small percent of the input sequences, the behavior did not appear in the execution sequence when it was in less than 50% of the input sequences. In addition, as mentioned earlier, in the execution phase the robot can collaborate with a human user, who may help the robot by performing some of the steps in its task. This human intervention may alter the path that the learner would have chosen to perform the task, while still obeying all the constraints of the training samples. Also, if the human user performs a behavior only partially the learner will need to complete it. When a human user performs an incomplete behavior, the number of times a behavior was repeated may not match a seen training example but it will obey the rules represented in the training examples. It is important to note that the human user can perform any behavior in the execution sequence but the following examples show only the cases where user input had a significant impact on the final execution sequence. In all of the examples the numbers in the parenthesis represent the parameter values for each behavior; the units can be found in the list above. We tested eight different scenarios, which cover all of the generalization cases presented in Section 3. These examples and the results are presented below.

**Scenario 1.** This constitutes an example where the training sequences were identical and it shows that the system can correctly encode tasks from generalization 1.a:

- *F*(2)*Sa*(1) *BSo*(1)*B*(2)*BSu*(1)*Su*(0.5)*V*(1) *E*(2) *CC*(2)
- *F*(2)*Sa*(1) *BSo*(1)*B*(2) *BSu*(1)*Su*(0.5)*V*(1) *E*(2) *CC*(2)
- *F*(2) *Sa*(1)*BSo*(1)*B*(2) *BSu*(1)*Su*(0.5)*V*(1) *E*(2) *CC*(2)

There is no variation in the training examples and thus the execution sequence produced by the system is identical to the training examples:

- *F*(2)*Sa*(1) *BSo*(1) *B*(2) *BSu*(1) *Su*(0.5) *V*(1) *E*(2) *CC*(2)

In this scenario, the helper performs the behavior add one egg which does not fully achieve the postconditions of that behavior.Our program is able to perform the behavior again to create an execution sequence consistent with the training examples; the execution sequence looks as follows:

- *F*(2)*Sa*(1) *BSo*(1)*B*(2) *BSu*(1) *Su*(0.5) *V*(1) *E*(1) *E*(1) *CC*(2)

**Scenario 2.** This scenario aims to illustrate that the system can identify and encode the type of generalizations described in 1.b:

*Example 1*. In a first setup, the learner is provided with the following demonstrations:

- *F*(2) *Sa*(1) *BSo*(1) *B*(2) *BSu*(1) *Su*(0.5)*V*(1) *E*(2) *CC*(2)
- *CC*(2)*V*(1) *F*(2) *BSu*(1) *E*(2) *Su*(0.5) *B*(2) *Sa*(1) *BSo*(1)
- *BSo*(1) *E*(2)*BSu*(1) *V*(1) *Su*(0.5)*Sa*(1)*CC*(2) *B*(2)*F*(2)

In this scenario the training examples were all different, with very few ordering constraints that are consistent throughout all the examples. The system

produces an execution sequence similar to one of the training examples (in this case, the third):

- *BSo*(1) *E*(2)*BSu*(1) *V*(1) *Su*(0.5) *Sa*(1) *CC*(2) *B*(2) *F*(2)

If in this example a helper performs a behavior at the beginning of the task, then the robot may take a different path to finish the task. The execution sequence above was chosen at random but if the helper performs the behavior *F* first, then the following sequence will be executed:

- *F*(2) *Sa*(1) *BSo*(1)*B*(2) *BSu*(1)*Su*(0.5)*V*(1)*E*(2) *CC*(2)

*Example 2.* In a first setup, the learner is provided with the following demonstrations:

- *BSo*(1)*Sa*(1) *Su*(0.5) *CC*(2) *E*(2)*F*(2) *BSu*(1)*B*(2) *V*(1)
- *V*(1) *F*(2) *B*(2) *Su*(0.5) *E*(2) *CC*(2) *Sa*(1) *BSu*(1) *BSo*(1)
- *B*(2)*E*(2) *CC*(2) *Sa*(1) *BSo*(1)*BSu*(1)*F*(2)*V*(1)*Su*(0.5)
- *E*(2)*F*(2) *Su*(0.5)*BSu*(1)*CC*(2) *V*(1)*Sa*(1) *B*(2)*BSo*(1)
- *B*(2)*E*(2)*CC*(2)*BSo*(1) *Sa*(1) *Su*(0.5) *F*(2)*V*(1) *BSu*(1)

In this scenario the training examples were all different, with no ordering constraints that are consistent throughout all the examples. There were two cases that started with the same three steps and thus the robot choose to start with those three steps since they had occurred the most in those positions. At this point in the program, the approach will randomly choose between *Sa* and *BSo*. In this example the helper assisted the robot after the first three steps, by performing *BSo*. Our approach was able to successfully find the next step that obeyed all the rules of the training data.

- *B*(2)*E*(2) *CC*(2)*BSo*(1) *Sa*(1) *Su*(0.5) *F*(2)*V*(1) *BSu*(1)

**Scenario 3.** This scenario falls under the generalization problem 1.c.i:

- *CC*(2)*Sa*(1) *V*(1) *Su*(0.5)*BSu*(1) *B*(2)*BSo*(1)*E*(2)*F*(2)
- *BSo*(1) *F*(2)*BSu*(1) *Sa*(1)*CC*(2)*E*(2) *B*(2)*Su*(0.5) *V*(1)
- *Su*(0.5) *V*(1)*B*(2) *CC*(2) *Sa*(1)*F*(2)*BSu*(1) *E*(2)*BSo*(1)

The above scenario is representative of cases in which a behavior needs to be executed before another behavior but not necessarily immediately before. In the training sequences above, add salt (*Sa*) always came before add eggs (*E*) but never right before it. The system produces an execution sequence similar to one of the training examples (in this case, the first):

- *CC*(2)*Sa*(1) *V*(1) *Su*(0.5) *BSu*(1)*B*(2)*BSo*(1)*E*(2)*F*(2)

**Scenario 4.** This scenario aims to show that the system can identify and encode the types of situations illustrated by generalization category 1.c.ii:

- *CC*(2)*F*(2) *Sa*(1) *BSo*(1) *BSu*(1) *B*(2) *E*(2) *V*(1) *Su*(0.5)
- *F*(2)*Sa*(1)*BSo*(1)*BSu*(1) *Su*(0.5)*CC*(2)*B*(2)*E*(2)*V*(1)
- *B*(2)*E*(2) *V*(1) *Su*(0.5) *CC*(2) *F*(2)*Sa*(1) *BSo*(1)*BSu*(1)

In this set of training examples, *F* always came before *Sa* which always came before *BSo* and *BSo* always came before *BSu*. Also *B* always came before *E* which always came before *V*. Fig. 5 shows the precondition graphs for two of the behaviors, one is a generalized behavior, *F+Sa+BSo+BSu*, and *CC*. The system

produces an execution sequence similar to one of the training examples (in this case, the third):

- *B*(2)*E*(2) *V*(1)*Su*(0.5)*CC*(2)*F*(2)*Sa*(1) *BSo*(1) *BSu*(1)

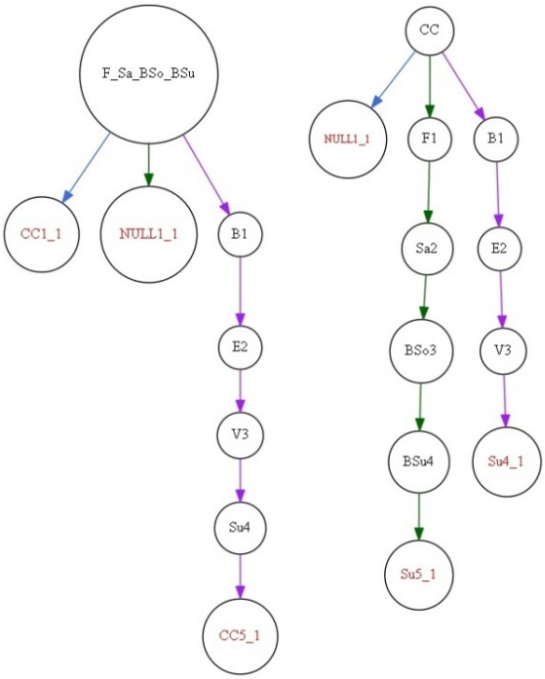


Fig. 5. Preconditions for Scenario 4 (behaviors *B*+*E*+*V*, *F*+*Sa*+*BSo*+*BSu*, and *CC*)

**Scenario 5.** This scenario aims to demonstrate that the system can successfully generalize problems of category 2:

*Example 1.* In a first setup, the learner is provided with the following demonstrations:

- *E*(2) *B*(2) *Su*(0.5) *BSu*(1) *F*(2)*BSo*(1) *N*(2) *Sa*(1) *V*(1) *CC*(2)
- *Sa*(1) *Su*(0.5) *E*(2) *CC*(2) *BSo*(1) *F*(2) *V*(1) *B*(2)*BSu*(1)
- *BSo*(1) *BSu*(1) *B*(2) *CC*(2) *E*(2)*F*(2)*Sa*(1) *Su*(0.5) *V*(1)

In this example, behavior *N* occurs in only one of the three sequences;this is lower than our threshold. Therefore *N*'s graph is removed from the set of possible behaviors and is not present in the execution sequence produced by the system:

- *E*(2) *B*(2) *Su*(0.5) *BSu*(1) *F*(2) *BSo*(1) *Sa*(1) *V*(1) *CC*(2)

*Example 2.* In a second setup, the learner is provided with these demonstrations:

- *E*(2)*B*(2) *Su*(0.5) *BSu*(1)*F*(2)*BSo*(1) *N*(2) *Sa*(1)*V*(1) *CC*(2)
- *Sa*(1)*Su*(0.5) *E*(2)*CC*(2) *BSo*(1)*F*(2)*V*(1) *B*(2) *BSu*(1)
- *BSo*(1) *N*(2) *BSu*(1) *B*(2)*CC*(2) *E*(2) *F*(2) *Sa*(1)*Su*(0.5) *V*(1)

In this example, behavior *N* appeared in two of the three training examples, and thus it is included as shown by the execution sequence being generated:

- *BSo*(1) *N*(2) *BSu*(1) *B*(2)*CC*(2)*E*(2)*F*(2)*Sa*(1) *Su*(0.5) *V*(1)

**Scenario 6.** This scenario aims to show that the system can identify and encode the types of situations illustrated by generalization category 1.c.iii:

- *CC*(2)*BSu*(1)*BSo*(1) *F*(2) *B*(2) *Su*(0.5) *Sa*(1) *V*(1) *E*(2)
- *V*(1)*CC*(2) *F*(2)*Sa*(1)*B*(2) *BSu*(1)*E*(2)*Su*(0.5)*BSo*(1)
- *Sa*(1) *Su*(0.5)*E*(2) *V*(1) *B*(2)*BSo*(1) *F*(2)*BSu*(1)*CC*(2)

In this case, the behavior add butter (*B*) was executed the fourth in each sequence in the example producing an execution sequence consistent with theexamples:

- *Sa*(1)*Su*(0.5) *E*(2)*V*(1)*B*(2) *BSo*(1) *F*(2) *BSu*(1) *CC*(2)

**Scenario 7.** This scenario aims to demonstrate that the system can successfully generalize problems of category 3:

*Example 1.* In the first setup, the learner is provided with these demonstrations:

- *Su*(0.5)*Sa*(1)*BSo*(1) *F*(2)*CC*(2)*St*(1) *St*(2)*St*(1) *St*(1)*BSu*(1) *V*(1)*B*(2)*E*(2) *St*(3)*St*(2)
- *BSu*(1)*BSo*(1) *E*(2) *Sa*(1) *CC*(2) *St*(1) *St*(1) *St*(1) *St*(1) *St*(1) *V*(1) *F*(2) *B*(2)*Su*(0.5) *St*(4) *St*(1)
- *B*(2)*V*(1)*E*(2)*BSu*(1) *Su*(0.5)*St*(2) *St*(2) *St*(1) *F*(2) *BSo*(1) *CC*(2) *Sa*(1) *St*(2) *St*(2) *St*(1)

Behavior *St* was repeated five times in two different places. Fig. 6 shows the precondition graph for *St*. The system detected the repeated behavior and at execution time it produced a sequence of *St* that used the most efficient way to stir 5 times:

- *BSu*(1)*BSo*(1) *E*(2) *Sa*(1) *CC*(2)*St*(4) *St*(1) *V*(1) *F*(2) *B*(2) *Su*(0.5) *St*(4) *St*(1)

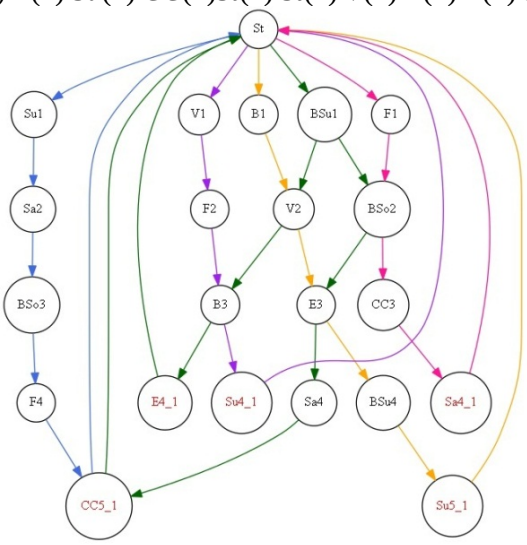


Fig. 6. Precondition Graph for Scenario 7 (Example 1: Behavior St)

*Example 2.* In the second setup, the learner is provided with these demonstrations:

- *Su*(0.5) *F*(1) *F*(1)*F*(0.5) *BSu*(1)*Sa*(1) *B*(2) *V*(1) *E*(2) *BSo*(1) *CC*(2)
- *E*(2) *BSo*(1)*Sa*(1)*B*(2)*F*(1)*F*(0.5)*F*(0.5)*F*(0.5)*BSu*(1)*V*(1) CC(2) Su(0.5)
- *CC*(2)*V*(1) *BSu*(1) *BSo*(1) *Su*(0.5) *E*(2) *F*(1)*F*(1)*F*(0.5) *Sa*(1)*B*(2)

In the second example the behavior *F* was repeated until there was 2.5 cups added in each sequence. The system detected the repeated behaviors and at execution time, produced a sequence whose post conditions met the training examples.

- *E*(2) *BSo*(1) *Sa*(1) *B*(2) *F*(2) *F*(0.5) *BSu*(1) *V*(1) *CC*(2) *Su*(0.5)

## 5. Discussion and future work

The experimental results presented above demonstrate that our system can successfully generalize the problems presented in Section 3. Furthermore the experiments establish that our system can successfully generate an execution sequence that is always consistent with the training samples.

However, our system could create a better execution sequence for the current environment if the state of the environment was recorded after a behavior was executed. This information would be used to figure out how many times a repeated behavior should occur.Along with this, future work also includes testing the proposed solution on a simulated and physical platform.

***Acknowledgements.*** The work was supported by the National Science Foundation under award IIS-0546876.

## References

1. Schaal, S. Is Imitation Learning the Route to Humanoid Robots? – Trends in Cognitive Sciences Vol. **6**, 1999, No 3, 242-323.
2. Voyles, R. M., J. D. Morrow, P. K. Khosla. Towards Gesture-Based Programming. – Robotics and Autonomous Systems, Vol. **22**, 1997, 361-375.
3. Voyles, R., P. Khosla. A Multi-Agent System for Programming Robots by Human Demonstration. – Integrated Computer-Aided Engineering, Vol. **8**, 2001, No 1, 59-67.
4. Pook, P. K., D. H. Ballard. Recognizing Teleoperated Manipulations. – In: Proc. Intl. Conf. on Robotics and Automation, 1993, 578-585.
5. Tominaga, H., J. Takamatsu, K. Ogawara, H, Kimura, K. Ikeuchi. Symbolic Representation of Trajectories for Skill Generation . – In: Proc., Intl. Conf. on Robotics and Automation, 2000, 4077-4082.
6. Ikeuchi, K., M. Kawade, T. Suehiro. Assembly Task Recognition with Planar, Curved and Mechanical Contacts. – In: Proc. IEEE Intl. Conf. on Robotics and Automation, 1993.
7. Kuniyoshi, Y., H. Inoue. Qualitative Recognition of Ongoing Human Action Sequences. – In: Proc., Intl. Joint Conf. on Artificial Intelligence, 1993, 1600-1609.
8. Avrahami-Zilberbrand, D., G. A. Kaminka. Fast and Complete Symbolic Plan Recognition. – In: Proc. Intl. Joint Conf. on Artificial Intelligence, 2005.
9. Abbas, T., B. A.MacDonald. Generalizing Topological Task Graphs From Multiple Symbolic Demonstrations in Programming by Demonstrations (PbD) Processes. – In: Proc. IEEE Intl. Conf. on Robotics and Automation, 2011, 3816-3821.
10. Konidaris, G., R. Grupen, A. Barto, S. Kuindersma. Robot Learning from Demonstration by Constructing Skill Trees. – The Intl. Journal of Robotics Research, Vol. **31**, 2012, 360-375.