Enabling Fast Instruction-Based Modification of Learned Robot Skills

Tyler Frasca[1], Bradley Oosterveld[2], Meia Chita-Tegmark[1], and Matthias Scheutz[1,2]
[1] Human-Robot Interaction Laboratory, Tufts University, 200 Boston Ave, Medford, MA 02155
[2] Thinking Robots, Inc., 12 Channel St STE 202, Boston, MA 02210
tyler.frasca@tufts.edu, brad@thinkingrobots.ai, mihaela.chita_tegmark@tufts.edu, matthias.scheutz@tufts.edu

Abstract

Much research effort in HRI has focused on how to enable robots to learn new skills from observations, demonstrations, and instructions. Less work, however, has focused on how skills can be *corrected* if they were learned incorrectly, *adapted* to changing circumstances, or *generalized/specialized* to different contexts. In this paper, a skill modification framework is introduced that allows users to modify a robot's stored skills quickly through instructions to (1) reduce inefficiencies, (2) fix errors, and (3) enable generalizations, all in a way for modified skills to be immediately available for task performance. A thorough evaluation of the implemented framework shows the operation of the algorithms integrated in a cognitive robotic architecture on different fully autonomous robots in various HRI case studies. An additional online HRI user study verifies that subjects prefer to quickly modify robot knowledge in the way we proposed in the framework.

Introduction

Consider a household robot that has learned how to prepare (regular) pancakes (e.g., from an online how-to website, Beetz et al. 2011) and already prepared them for its owner multiple times. One day, however, the owner decides to try out vegan pancakes and tells the robot to replace the regular milk with soy milk in the pancake recipe, rather than teaching the modified recipe from scratch.

Such adaptations of known recipes and processes are very natural for humans, but they are currently not easy to make for robots. For one, because they require a knowledge representation format where different components of a script representing a skill are accessible and can be replaced without negatively impacting other parts, or other scripts in the robot's knowledge base. For example, skill representations in terms of large weight vectors embedded in a neural network do not lend themselves to easy modification (even if one knew how each weight was related to each particular skill, changing a few of those weights to accommodate the modified skill might also modify other stored skills given the distributed data representations in most neural networks). Similarly, modifying an action for a particular state in a given action policy without modifying an RL agent's reward function under which the policy was generated could lead to the action being changed back if the agent continues to learn based on its existing reward function. Both types of action representation thus likely require complete retraining of all learned skills with the modified skill included.

To enable fast modification, we present a knowledge representation framework which allows a user to modify skills to take effect immediately without the need for relearning all existing skills. We accomplish this by explicitly representing all relevant aspects of a skill – its signature with arguments and parameters, and its action steps with and pre-, operating, and post-conditions – which then allows for quick modifications of any of these components through formal operators defined on them. Since all modifiable aspects are explicit and can be described in natural language, robots can narrate or visualize their knowledge in a way that is intuitive to humans, and humans can give robots verbal (or graphical) instructions to perform the necessary modifications which are mapped onto formal operators that perform the necessary changes to the scripts. We evaluated the implemented framework and demonstrated in four case studies in which a human instructs different robots to modify their known actions that the robots are able to accommodate and execute the modifications immediately. We also verified in a user study that people, indeed, want the option of instructing robots to modify their knowledge.

This paper thus makes the following contributions towards correcting, generalizing/specializing learned actions:

- a formal skill modification framework that allows users to modify a robot's stored skills through verbal or graphical instructions,
- an implementation and evaluation of the framework in a cognitive robotic architecture,
- four HRI case studies demonstrating the operation of the algorithms on different autonomous robots, and
- a user study showing if and to what extend human users would want to modify a robot's skills and in what form.

Related Work

Prior approaches to skill learning differ with respect to their required input modalities as well as their employed knowledge representations for the learned skills. While some methods are purely data-driven (e.g., RL methods), others are purely knowledge-based (e.g., some NL methods).

Roughly, we can distinguish between *how a skill is described* (e.g., from explicit teaching through demonstrations (Argall et al. 2009; Chernova and Thomaz 2014; Mohseni-Kabir, Chernova, and Rich 2014), instructions (Mohan and Laird 2011; Scheutz et al. 2017; Rybski et al. 2007), crowd-sourced datasets (Clair et al. 2016), or through unsupervised learning such as exploration (Baranes and Oudeyer 2013; Pape et al. 2012), observation (Andry et al. 2001), or trial-and-error (Mahadevan and Connell 1992; Whitehead and Ballard 1991)) from *how a skill is learned* (e.g., using different variants of reinforcement learning (Hershkowitz, MacGlashan, and Tellex 2015; Konidaris et al. 2010), backpropagation (Gu et al. 2017), explicit natural language instructions (Suddrey et al. 2017; Frasca et al. 2018), or gestures (Calinon and Billard 2007)), and furthermore *how the learned skill is represented* in the robotic architecture (e.g., as action scripts, behavior trees, action policies, Q values, or keyframe-based trajectories).

Regardless of how the skill was described and learned by the robot, the robot should be able to modify it subsequently should the need arise. Corrective demonstrations, for example, have been used for updating primitives through kinesthetic teaching. Akgun et al. (2012) introduced methods for learning keyframe-based models of skills and adding/removing keyframes in the learned model in subsequent interactions. Gutierrez et al. (2018) present an incremental task modification algorithm based on a demonstration that creates and probabilistically selects corrections for structural modifications in a finite-state machine task model. Gutierrez et al. (2019) build on this approach by providing a demonstration at or near execution failure so as to only having to locally update the model. The local update procedure determines if the demonstration represents a known primitive (as opposed to a new one), in which case the demonstration is added and the primitive retrained. However, it is unclear if a single demonstration significantly impacts the primitive model and may require multiple corrective demonstrations. In contrast, our approach works with a single instruction to modify the known skill. Additionally, their corrective demonstrations are limited to trajectories, whereas ours allows a user to specify other corrections including navigation directions and locations.

Repairing actions has also been used in formal logic systems, including linear temporal logic (Boteanu et al. 2017). When the system receives an instruction and is unable to synthesize a controller, it prompts a user with possible causes and environment assumptions that if true could allow the system to execute the action. Unlike their reactive only approach, our system allows the user to be both reactive, if the system cannot perform an action, and proactive when the user wants to modify the task. It is unclear if the repaired instruction is saved for later, whereas ours saves the modified action for future use and modification.

A Framework for Representing and Modifying Robot Actions

For the action modification framework, we assume explicit action representations such as action scripts to capture the robot's procedural knowledge, which allows a human to abstract over low level implementations. We thus start with a formal definition of "action script" and then introduce operators on action scripts that perform various types of modifications (these operators can easily be adapted to other formalisms like programs, behavior trees, plans, recipes, etc.).

Action Representation

Action scripts (e.g., Brick, Schermerhorn, and Scheutz 2007; Scheutz et al. 2007) are compact ways of specifying hierarchical robot behavior without explicitly modeling the entire robot or world state relative to each action. They are defined by an expression $\alpha(p_1 : t_1, p_2 : t_2, \ldots, p_m : t_m)$ where α is an action symbol and each $p_i : t_i$ is a parameter p_i of a given type t_i (e.g., a reference to a graspable object). Having typed parameters is important for generalizing actions. Action scripts are similar to STRIPS operator (Fikes and Nilsson 1971) in that each has a set of pre-conditions Π^α that need to be met for the action to be executable and a set of post-conditions Σ^α that will be true when the action succeeds. However, ours has a set of operating conditions Ω^α which must hold true during action execution for the execution to succeed (as otherwise the action will fail), and a set of failure post-conditions (or failure effects) Φ^α that will be true when action execution fails. All condition sets are finite, containing first-order formulas over a finite set of predicates. Semantically, pre-conditions determine an equivalence class of world states (without requiring representations of those states in the system) in a transition system such that when a given action is started in any of the equivalent states and the operating conditions hold true throughout the execution, the system will end up in a successor state which is a member of the equivalence class of states defined by the success post-conditions; otherwise it will end up in a state in the equivalence class of the failure post-conditions. Scripts contain a finite sequence of action steps $\alpha_1; \alpha_2; \ldots; \alpha_n$ (without additional non-action expressions such as control expressions like "if-then-else" conditions, "for" and "while" loops, event descriptions, observer expressions). Whereas STRIPS does not per say define sequence of operators. Each α_i represents an action script, or action primitive, where an action primitive is represented in the same manner, except it provides a single operation instead of containing a sequence of steps. [1]

[1] Action scripts are instances of programs in quantifier-free first-order dynamic logic with empty assignments, see (Harel 1979). Consequently, the meaning of dynamic logic formulas $\Pi^\alpha \rightarrow [\alpha]\Sigma^\alpha$ and $\Pi^\alpha \rightarrow \langle\alpha\rangle\Sigma^\alpha$ which state that if Π^α (the preconditions) are true, then all or one execution path of α, respectively, will make Σ^α (the post-conditions) true, can be transferred to action scripts as well. As a result, action scripts inherit the semantics of dynamic logic and with it provable properties such as action sequencing (and thus script composition): given $\Pi^\alpha \rightarrow \langle\alpha\rangle\Psi^\alpha$ and $\Psi^\alpha \rightarrow \langle\beta\rangle\Sigma^\alpha$, then $\Pi^\alpha \rightarrow \langle\alpha;\beta\rangle\Sigma^\alpha$ (cp. to Hoare's rule of composition (Hoare 1969)). The main differences between action scripts and standard dynamic logics is that operating conditions that hold true throughout the execution cannot be explicitly expressed in dynamic logic (one would have to use a more expressive temporal operators as in CTL or CTL*).

Action	
Name:	HandOver
Parameters	Agent: a1 Agent: a2 Object: o1
Pre-Conditions	holding(a1,o1)
Steps	locate(a1,a2) moveObj(a1,o1,towards(a2)) graspObj(a2,o1) releaseObj(a1,o1)
Operating-Conditions	holding(a1,o1)
Post-Conditions	not(holding(a1,o1)) holding(a2,o1)

Table 1: Example action script for the action HandOver.

Action Modifications

Given an action script $\alpha(p_1 : t_1, p_2 : t_2, \ldots, p_m : t_m)$ with pre-conditions Π^α, operating conditions Ω^α, post-conditions Σ^α and Φ^α, and with action steps $\alpha_1; \alpha_2; \ldots; \alpha_n$, where each α_i is either a primitive action or an action script with its own conditions Π^{α_i}, Ω^{α_i}, Σ^{α_i} and Φ^{α_i}, we want to allow for modifications to any aspect of α and possibly recursively, to any part of α_i. Modifications can *insert* or *delete* information such as adding or deleting conditions or action steps, or *replace* information such as altering existing conditions or action steps with different formulas. Accordingly, we define three types of modification operators, O_{ins}, O_{del}, and O_{rep}. O_{ins} and O_{del} which yield a modified action script described in the equations below (where "$Z[X/Y]$" means that Y replaces X in Z):

$$O_{ins}(\alpha, \gamma, k, \uparrow) := \alpha[\alpha_k/\gamma; \alpha_k] \quad (1)$$

$$O_{ins}(\alpha, \gamma, k, \downarrow) := \alpha[\alpha_k/\alpha_k; \gamma] \quad (2)$$

$$O_{ins}(\alpha, \gamma, k, \mathcal{C}) := \alpha[\mathcal{C}^{\alpha_k}/\mathcal{C}^{\alpha_k} \cup \{\gamma\}] \quad (3)$$

where k, the action step index received from the user input system, is $1 \leq k \leq n$ and $\mathcal{C} \in \{\Pi, \Omega, \Sigma, \Phi\}$.

Equation 1 and Equation 2 define the insertion of the new action γ either before or after the k-th action step, respectively, whereas Equation 3 defines the addition of γ, where γ is a new condition.

Similarly, Equation 4 and Equation 5 define the removal of action step β relative to the k-th action step, and Equation 6 defines the removal of a condition: since the conditions are sets, there is only a single instance of β, where β is the condition being removed.

$$O_{del}(\alpha, \beta, k, \uparrow) := \alpha[\beta; \alpha_k/\alpha_k] \quad (4)$$

$$O_{del}(\alpha, \beta, k, \downarrow) := \alpha[\alpha_k; \beta/\alpha_k] \quad (5)$$

$$O_{del}(\alpha, \beta, k, \mathcal{C}) := \alpha[\mathcal{C}^{\alpha_k}/\mathcal{C}^{\alpha_k} - \{\beta\}] \quad (6)$$

where $1 \leq k \leq n$ and $\mathcal{C} \in \{\Pi, \Omega, \Sigma, \Phi\}$.

Finally, the replace operator O_{rep} replaces the k-th action step or condition β with γ in any of the condition sets:

$$O_{rep}(\alpha, \gamma, k) := \alpha[\alpha_k/\gamma] \quad (7)$$

$$O_{rep}(\alpha, \gamma, \beta, k, \mathcal{C}) := \alpha[\mathcal{C}^{\alpha_k}/\mathcal{C}^{\alpha_k} - \{\gamma\} \cup \{\beta\}] \quad (8)$$

where $k \in \{\Pi, \Omega, \Sigma, \Phi\}$. It is easy to see that O_{rep} can be defined in terms of O_{ins} and O_{del}:

Corollary: Let β be the k-th action step of α. Then $O_{rep}(\alpha, \gamma, \beta, k) = O_{del}(O_{ins}(\alpha, \gamma, k, \uparrow), \beta, k, \downarrow)$. And similarly, $O_{rep}(\alpha, \gamma, \beta, k, \mathcal{C}) = O_{del}(O_{ins}(\alpha, \gamma, k, \mathcal{C}), \beta, k, \mathcal{C})$, where β is the condition in alpha.

All of the above operations can be proved to accomplish the intended effects on action script through induction on the structure of the script, e.g., given an action script α of the form $\alpha_1; \alpha_2; \ldots; \alpha_n$, then $O_{rep}(\alpha, \gamma, k)$ will result in the script $\alpha_1; \ldots; \alpha_{k-1}; \gamma; \alpha_{k+1}; \alpha_n$ (we omit the proofs for space reasons). Similarly, all of the above operators can be extended to recursive versions where insertions, deletions, and replacements occur not only in script action steps, but also in actions steps of subscripts, their subscripts, and so on (again, we omit the formal definitions for space reasons).

Evaluation of the Framework Implemented in a Cognitive Robotic Architecture

The above framework can be directly implemented in any robotic architecture that has some sort of action sequencer, i.e., a component or algorithm that takes in action scripts, recipes, behavior trees, plans, etc. and executes them step by step (e.g., see Kortenkamp and Simmons 2008). The details of the implementation will vary based on the architecture (e.g., what particular script representation the architecture uses, where scripts are stored, etc.) and how the architecture allows for triggering modifications (e.g., modifications might come in as task goals, or via user interfaces such as GUIs or spoken dialogue, Nirenburg et al. 2018). We implemented the framework in the DIARC architecture (Scheutz et al. 2019) due to the easy access of action scripts in its action database and its integrated and customizable natural language processing system which allows for script-access through spoken natural language instructions (additionally, DIARC allows for learning of new action scripts from instructions and can verbally describe scripts). We also made the particular implementation decision (that can be changed in different implementations) that when an action step α_k matching a modification argument is found, the first matching instance is always selected even if there are multiple possibilities (if no matching action is found, then the modification operation returns without updating the action database). A copy of the retrieved action representation is created to allow for the evaluation of the effect of the modification before updating the action database. Upon completion of the modification, if the new signature, pre-conditions, or effects are different from the original action, the system retains the original action script, hence the modifications will not affect the rest of the knowledge database, otherwise the system updates the existing action script, overwriting the original representation.

We evaluated the correctness of the framework implementation on 50 randomly generated action scripts as follows. For each randomly generated action script, we randomly generated 1 to 4 parameters, and 1 to 4 conditions of ran-

Action	
Name	action_0
Parameters	Agent: actor Object: obj_0 Object: obj_1
Pre-Conditions	holding(actor, mug) on(obj_0, obj_1)
Steps	moveObj(actor, obj_0, down) placeOn(tyler, ball, mug) graspObj(dempster, obj_1)
Operating-Conditions	
Post-Conditions	on(obj_1, table)

Table 2: Randomly generated action using the arguments, initial actions, and conditions

domly selected type $\{\Pi, \Omega, \Sigma, \Phi\}$ using

Parameter types and arguments:
Thing = {Agent,Object}
Agent = {andy,tyler,dempster,laura}
Object = {ball,mug,table,cup}
Direction = {right,left,up,down}

Conditions:
touching(Thing,Thing), on(Object,Object)
holding(Agent,Object), see(Agent,Object)

And we recursively created new action steps for the script starting with the following set of initial actions:

Initial Actions and Type Signatures:
graspObj(Agent,Object)
moveObj(Agent,Object, Direction)
pickUp(Agent,Object)
placeOn(Agent,Object,Object)
releaseObj(Agent,Object)

and then added each newly created action to the set of actions to choose from (call it $Actions$), so that subsequently generated actions would choose from the initial primitive actions as well as more complex actions. For each action in $Actions$ we then created nine random modifications as follows: we either operated on one of the steps α_k within the action, or on one of its four conditions $\{\Pi^\alpha, \Omega^\alpha, \Sigma^\alpha, \Phi^\alpha\}$.

For insertion and replacement, we randomly picked an action from $Actions$ for a new action step and a random condition from $Conditions$ for condition operations (deletions for both actions and conditions simply removed the actions and formula, respectively).

We manually compared the results of 450 automatically modified action scripts to those modified by hand and verified they all matched up. The perfect result confirmed the correctness of the implementation and integration of the framework into the robotic DIARC architecture. It allowed us to further investigate in human-robot interaction case studies how the framework can be used to make quick modifications through natural language dialogues.

Validation on Fully Autonomous Robots

We also validated the fully functional implementation of the action modification framework in four case studies on a PR2 and a Nao robot, where the only difference in the configurations was the set of primitive actions each robot is capable of performing (note that even though a robot may not be able to perform an action, it can still reason about it and describe it). We tested the following key capabilities: inserting, removing, and replacing action steps and adding conditions, showing that our approach works in realistic world problems where a human needs to make modify the robots skills. In each case, a human instructor, Brad, instructs a robot using a restricted set of language utterances to perform actions and also provides modification descriptions for one of the actions. We show transcripts of the interactions as well as the generated goal predicates (for details, see Scheutz et al. 2017), the script representation of the modified action at the start of the interaction, the updated representation resulting from the modification, and discuss the importance of each modification. Note, although we used NL to provide the instructions, a GUI could also be used where the user selects the action to modify and how to modify it.

Generalizing an Action by Replacing a Step

This case shows the replacement of a problematic step in a known script. The interaction begins by Brad asking the robot to pass him a knife.

Brad: Pass me the knife.
pass(self,Brad,knife)
Robot: Ok

The robot performs the pass action shown in the First row of Figure 1, which results in it dropping the knife out of Brad's reach. Since Brad is familiar with the action script used for the pass action, he can directly provide a modification that generalizes it to passing a knife to a person in any position, not just one in front of it.

Brad: When you pass me the knife replace move the knife forward with move the knife toward me.
modifyAction(pass(self,brad,knife), replace(moveObj(self,knife, toward,brad), moveObj(self,knife,forward)), step())
Robot: Ok.

Original Script	New Script
Name: pass	**Name:** pass
Parameters: act1 act2 obj	**Parameters:** act1 act2 obj
Steps:	**Steps:**
findObj act1 obj	findObj act1 obj
graspObj act1 obj	graspObj act1 obj
moveObj act1 obj up	moveObj act1 obj up
moveObj act1 obj forward	**moveObj act1 obj toward act2**
releaseObj act1 obj	releaseObj act1 obj

Discussion The robot already knows how to perform the instructed action, but it requires the person receiving the object to be directly in front of it. As this will not always be the case, Brad provides a generalizing modification which instantly makes it applicable in more situations.

Repairing an Action by Inserting a Missing Step

Suppose the robot learned the squat action described below. Brad instructs the robot to perform the action and recognizes an omitted step. Upon recognizing the error, Brad asks the

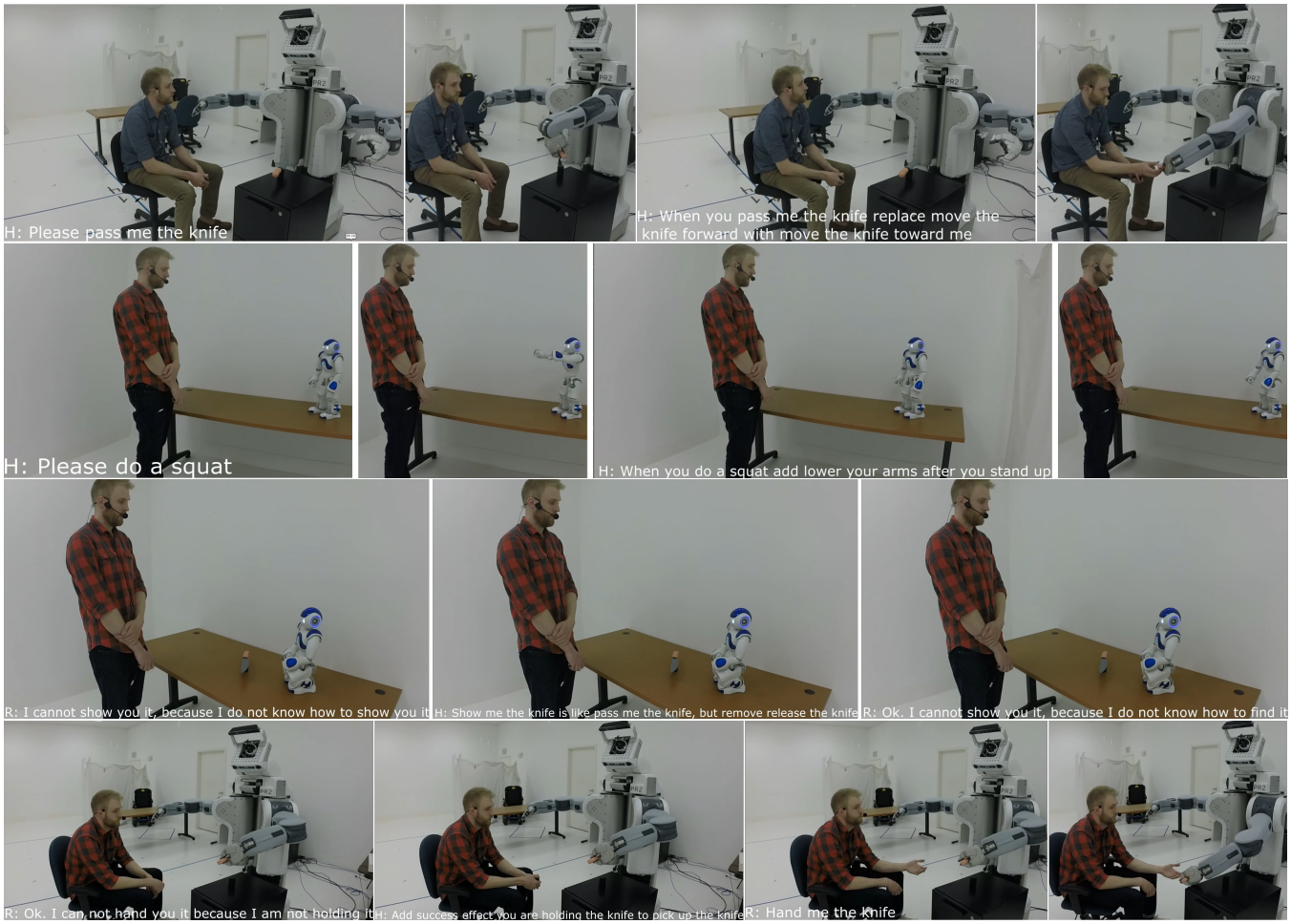

Figure 1: First row: When passing the knife the robot moves the knife forward, Brad tells it to move the knife toward him instead. Second row: The robot performs the squat action and Brad notices it didn't lower its arms. He repairs the action by telling the robot to lower its arms after standing up. Third row: The robot doesn't know show the knife. Brad tells the robot that show is similar to passing a knife, however don't release it. Fourth row: Asking the robot to hand over the object, which the robot knows how to perform, but it requires that it holding the knife. The robot doesn't understand that it is holding the knife when it picks it up. Brad instructs the robot that it is indeed holding the knife.

robot to describe the learned action and uses that information to instruct a fix as shown in the Second row of Figure 1.

Brad: Describe how to do a squat.
describeAction(self,squat)

Robot: To do a squat I raise my arms, I crouch down, and I stand up.

Brad: When you do a squat make sure you lower your arms after you stand up.
modifyAction(squat(self), insert(lower(self,arms)), step(after(standUp(self))))

Robot: Ok.

Discussion Initially, the robot does not know how to perform a squat. Brad provides instructions, but forgets to tell the robot to lower its arms (omissions that happen typically when the human has more domain knowledge than the robot). Additionally, the human could have knowledge of operating assumptions the robot is unaware are important.

Original Script	New Script
Name: squat	**Name:** squat
Parameters: act	**Parameters:** act
Steps:	**Steps:**
raise act arms	raise act
crouch act	crouch act
standUp act	standUp act
	lower act arms

When Brad notices the robot does not lower its arms after squatting down, instead of having to reteach the robot, he simply adds the missing action.

Generating an Action from an Existing Script

Here, Brad teaches the robot an action similar to a known one by describing it in terms of the known action as seen in the Third row of Figure 1. The newly generated action is

stored independently of the action on which it was based.

Brad: Show me the knife.
show(self,brad,knife)
Robot: Sorry, I do not know how to show you the knife.
Brad: Show me the knife is like pass me the knife without release the knife.
modifyAction(show(self,brad,knife), pass(self,brad,knife), remove(release(self,knife)), step())
Robot: Ok.
Brad: Show me the knife.
show(self,brad,knife)
Robot: I cannot show you it, because I do not know how to find it.
Brad: Describe how to show me the knife.
describeAction(self,show(knife))
Robot: To show you the knife, I find it, grab it, I move it up, and I move it toward you.

Original Script	New Script
Name: pass	**Name:** show
Parameters: act1 act2 obj	**Parameters:** act1 act2 obj
Steps:	**Steps:**
findObj act1 obj	findObj act1 obj
graspObj act1 obj	graspObj act1 obj
moveObj act1 obj up	moveObj act1 obj up
moveObj act1 obj toward act2	moveObj act1 toward act2
releaseObj act1 obj	

Discussion This scenario is similar to that of making vegan pancakes when the robot already knows how to make regular pancakes. Instead of teaching the new action from scratch, Brad modifies an existing skill the robot knows how to perform. Additionally, note that when Brad asks the robot to perform the action it replies that it can not do so because it cannot physically perform some of the primitive actions (i.e., detecting knife given its limited vision system). Yet, as evidenced by its description, the robot has successfully learned the modification.

Adding an Effect to an Action Script

Here, Brad provides the robot additional semantic information, the post-condition, for one of its actions which makes the action usable by an automated planner as seen in the Fourth row of Figure 1.

Brad: Pick up the knife.
pickUp(self,knife)
Robot: Ok.
Brad: Hand me the knife.
handOver(self,knife)
Robot: Sorry, I cannot hand you the knife because I am not holding the knife.

The *handOver* action has the precondition that the robot must beholding the object it is to hand over. The pick up action that the robot performs does satisfy this condition, but the robot is not aware of that. The human modifies the pick up action so that the robot is aware of its effect on the state of the world.

Brad: After you pickup the object successfully you are holding the object.
modifyAction(pickUp(self,knife), insert(holding(self,knife)),post-condition(success))
Robot: Ok.

Original Script	New Script
Name: pickUp	**Name:** pickUp
Parameters: act1 obj	**Parameters:** act1 obj
Effects:	**Effects: holding(act1, obj)**
Steps:	**Steps:**
findObj act1 obj	findObj act1 obj
graspObj act1 obj	graspObj act1 obj
moveObj act1 obj up	moveObj act1 obj up

Discussion When learning a new task, the robot may not know how it completely affects the state of the world. Although the robot may observe the environment and learn some effects, it may not acquire all knowledge about the action's impact. For example, if a human teaches the grasp action and doesn't inform the robot that when it grasps an object the robot is then holding it. Therefore, the robot should be able to get additional conditional information from instructors. This scenario demonstrates the robots ability to quickly gain the knowledge required to hand over the knife.

Human-Subjects Study

To better understand how users would experience the act of modifying robot behaviors, we conducted an on-line study, in which participants had to determine how to modify different robot behaviors and then answer questions about their impressions of the modification process.

Subjects

Participants were recruited on the Amazon Mechanical Turk platform. Thirty-one participants completed the study, 14 female, 16 male and one did not identify their gender with ages ranging from 24 to 67 (Mean = 43.7 years, SD = 11.4).

Procedure

Participants watched three short videos in which robots end up incorrectly performing a task: the "passing the knife" task performed by the Willow Garage PR2 robot as described in detail above, and two additional tasks, "setting the table" and "clearing the plate", preformed by the Kinova Gen 3 robotic arm. In the "setting the table" task, the robot fails by placing two forks on top of each other, next to the plate. In the "clearing the plate" task the robot fails by not removing the fork and the knife from the plate before emptying the plate's contents in the trash, resulting in throwing away the cutlery. After watching the videos, the participants were asked to reflect on how they would modify the robot's behavior, first by formulating and writing down the instructions they would give to the robot to correct its behavior, and then, after watching the video one more time, by choosing a modification from a list of multiple choices. We then asked participants how they would like to convey these instructions to the robot: by using a programming language, by using a

Item	PR2		Kinova	
	Agree	Rating	Agree	Rating
I feel comfortable making a modification to the robot's behavior.	73%	2.81 ± 0.21	93%	3.33 ± 0.17
I would ask the robot to explain the modification in order to assess its correctness.	67%	2.63 ± 0.20	73%	2.80 ± 0.24
I would like the robot to make autonomous self-improvements to its behavior.	57%	2.30 ± 0.24	60%	2.67 ± 0.28
I feel more comfortable knowing I can modify the robot's behavior.	93%	3.47 ± 0.13	97%	3.63 ± 0.10
I would trust the robot more knowing I could modify its behavior.	97%	3.57 ± 0.10	93%	3.50 ± 0.11
I would like the robot to provide feedback if the modification might impact other behaviors.	90%	3.33 ± 0.16	93%	3.40 ± 0.17
I would ask the robot to explain its behavior.	63%	2.70 ± 0.20	66%	2.77 ± 0.22

Table 3: Impressions on the modification of robot behavior: percentages of people who agreed or strongly agreed with the statements, and mean Likert-scale ratings with standard errors.

graphical user interface (GUI), or by speaking to the robot using natural language. We did not include demonstrations because they are not always applicable nor practical. Finally, to capture participants' impressions regarding the capability of modifying behavior in robots we asked them to rate a series of statements (see Table 3) on a 5-point Likert scale from "strongly disagree" (1) to "strongly agree" (5) for each of the two robots they saw performing the task.

Results

Participants predominantly wanted to convey behavior modification instructions to the robot using natural language: 67% of participants wanted to use natural language for modifying the robot's behavior for the passing the knife task (23% wanted to use a programming language, and 10% a GUI), 53% for the setting the table task (30% programming language, 17% GUI), and 67% for the clearing the plate task (23% programming language, 10% GUI). Almost all participants correctly identified the modification needed from a set of multiple choices: 77% for the passing the knife scenario, 83% for the setting the table scenario, and 93% for the clearing the plate scenario. Participants indicated that they felt comfortable knowing they could modify the robot's behavior, and also that they felt comfortable making the modification: 93% and 73% agreed and strongly agreed with these statements respectively for the PR2 robot, and 97% and 93% for the Kinova Gen 3 robotic arm. Also, participants indicated that they would trust the robot more knowing they could modify its behavior: 93% agreed or strongly agreed to this statement for the PR2 robot and 97% for the Kinova Gen 3 robotic arm. To note that a much lower percentage of people wanted the robot to make autonomous self-improvements to its behavior (PR2: 57%, Kinova: 60%), suggesting that people appreciate the capability of choosing and conveying their own modifications to the robot.

Discussion and Future Work

The user study demonstrates the need for modifying robot knowledge quickly, ideally through verbal instructions (but possibly also through other means such as GUIs). While more than half of the subjects would like robots to make self-improvements to their behaviors, almost all users want to have the option of modifying the robots' behaviors themselves and would also trust robots that allowed such user-instructed modifications more. These results show the need of an action modification framework that can be easily controlled by users, which our framework thus addresses.

While the framework can be integrated into different robotic architectures as long as they provide explicit representations of all relevant aspects of the robot's procedural knowledge, our particular integration into an architecture with a natural language interface allows non-expert users to naturally modify the robot's skills, inserting, removing, and replacing action steps, pre-conditions, operating conditions, and post-conditions.

One interesting question for future work is how to handle modified scripts that are used as subscripts in other scripts where the modification might affect the proper operation of the script. For example, if the initial squat script where the robot did not lower its arms was part of a larger exercise of doing repeated squats with the hands up, the modified script would lead to the wrong execution. This kind of inconsistency is difficult to spot unless the post-conditions of the modification break the preconditions of the next action step in the larger script (see also (Nirenburg et al. 2018)). Future work will investigate the extent to which consistencies can be detected and resolved automatically (e.g., by renaming variables, conditions, and script names, and duplicating parts of scripts) or need to be resolved in consultation with the user (e.g., via dialogue interactions about the actions and scripts implicated by the modification).

Conclusion

We introduced a skill modification framework that allows users to make different types of modifications to a robot's stored skills through instruction. We formally presented the operations, implemented them in a robotic architecture to enable skill modifications, and thoroughly evaluated the integration using random sampling from a large space of possible scripts and modifications that were manually verified. Through four case studies, we demonstrated the operation of the integrated algorithms in a robotic architecture run on two different robots. A user study showed that their trust in robots providing action modification was higher, and that natural language was the preferred modification method.

Acknowledgments

This work was supported in part by U.S. Office of Naval Research grants #N00014-18-1-2503 and #N00014-19-1-2311.

References

Akgun, B.; Cakmak, M.; Yoo, J. W.; and Thomaz, A. L. 2012. Trajectories and keyframes for kinesthetic teaching: A human-robot interaction perspective. In *Proceedings of the seventh annual ACM/IEEE international conference on Human-Robot Interaction*, 391–398.

Andry, P.; Gaussier, P.; Moga, S.; Banquet, J. P.; and Nadel, J. 2001. Learning and communication via imitation: an autonomous robot perspective. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans* 31(5): 431–442.

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.

Baranes, A.; and Oudeyer, P.-Y. 2013. Active learning of inverse models with intrinsically motivated goal exploration in robots. *Robotics and Autonomous Systems* 61(1): 49 – 73.

Beetz, M.; Klank, U.; Kresse, I.; Maldonado, A.; Mösenlechner, L.; Pangercic, D.; Rühr, T.; and Tenorth, M. 2011. Robotic Roommates Making Pancakes. In *11th IEEE-RAS International Conference on Humanoid Robots*.

Boteanu, A.; Arkin, J.; Patki, S.; Howard, T.; and Kress-Gazit, H. 2017. Robot-initiated specification repair through grounded language interaction. *arXiv preprint arXiv:1710.01417* .

Brick, T.; Schermerhorn, P.; and Scheutz, M. 2007. Speech and Action: Integration of Action and Language for Mobile Robots. In *Proceedings of the 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1423–1428. San Diego, CA.

Calinon, S.; and Billard, A. 2007. Incremental Learning of Gestures by Imitation in a Humanoid Robot. In *Proceedings of the ACM/IEEE International Conference on Human-robot Interaction*, HRI '07, 255–262. New York, NY, USA: ACM.

Chernova, S.; and Thomaz, A. L. 2014. Robot learning from human teachers. *Synthesis Lectures on Artificial Intelligence and Machine Learning* 8(3): 1–121.

Clair, A. S.; Saldanha, C.; Boteanu, A.; and Chernova, S. 2016. Interactive hierarchical task learning via crowdsourcing for robot adaptability. In *Refereed workshop Planning for Human-Robot Interaction: Shared Autonomy and Collaborative Robotics at Robotics: Science and Systems, Ann Arbor, Michigan. RSS*.

Fikes, R. E.; and Nilsson, N. J. 1971. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2(3): 189 – 208.

Frasca, T.; Oosterveld, B.; Krause, E.; and Scheutz, M. 2018. One-Shot Interaction Learning from Natural Language Instruction and Demonstration. In *Proceedings of the Sixth Annual Conference on Advances in Cognitive Systems*. Stanford, USA: ACS.

Gu, S.; Holly, E.; Lillicrap, T.; and Levine, S. 2017. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 3389–3396.

Gutierrez, R. A.; Chu, V.; Thomaz, A. L.; and Niekum, S. 2018. Incremental task modification via corrective demonstrations. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 1126–1133. IEEE.

Gutierrez, R. A.; Short, E. S.; Niekum, S.; and Thomaz, A. L. 2019. Learning from corrective demonstrations. In *2019 14th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 712–714. IEEE.

Harel, D. 1979. *First-order Dynamic Logic*. Lecture Notes in Computer Science. Springer-Verlag.

Hershkowitz, D. E.; MacGlashan, J.; and Tellex, S. 2015. Learning propositional functions for planning and reinforcement learning. In *2015 AAAI Fall Symposium Series*.

Hoare, C. 1969. An axiomatic basis for computer programming. *Communications of the ACM* 12(10): 576–580.

Konidaris, G.; Kuindersma, S.; Grupen, R.; and Barto, A. G. 2010. Constructing Skill Trees for Reinforcement Learning Agents from Demonstration Trajectories. In Lafferty, J. D.; Williams, C. K. I.; Shawe-Taylor, J.; Zemel, R. S.; and Culotta, A., eds., *Advances in Neural Information Processing Systems 23*, 1162–1170. Curran Associates, Inc.

Kortenkamp, D.; and Simmons, R. 2008. Robotic Systems Architectures and Programming. In *Handbook of Robotics*, 187–206. Springer.

Mahadevan, S.; and Connell, J. 1992. Automatic programming of behavior-based robots using reinforcement learning. *Artificial Intelligence* 55(2): 311 – 365.

Mohan, S.; and Laird, J. E. 2011. Towards Situated, Interactive, Instructable Agents in a Cognitive Architecture. In *AAAI Fall Symposium: Advances in Cognitive Systems*.

Mohseni-Kabir, A.; Chernova, S.; and Rich, C. 2014. Collaborative learning of hierarchical task networks from demonstration and instruction. In *RSS Workshop on Human-Robot Collaboration for Industrial Manufacturing, Berkeley, CA*.

Nirenburg, S.; McShane, M.; Beale, S.; Wood, P.; Scassellati, B.; Magnin, O.; and Roncone, A. 2018. Toward Human-Like Robot Learning. In Silberztein, M.; Atigui, F.; Kornyshova, E.; M'etais, E.; and Meziane, F., eds., *Natural Language Processing and Information Systems*, volume 10859 of *Lecture Notes in Computer Science*, 73–82. Springer.

Pape, L.; Oddo, C. M.; Controzzi, M.; Cipriani, C.; Förster, A.; Carrozza, M. C.; and Schmidhuber, J. 2012. Learning tactile skills through curious exploration. *Frontiers in Neurorobotics* 6: 6.

Rybski, P. E.; Yoon, K.; Stolarz, J.; and Veloso, M. M. 2007. Interactive robot task training through dialog and demonstration. In *Proceedings of the ACM/IEEE international conference on Human-robot interaction*, 49–56. ACM.

Scheutz, M.; Krause, E.; Oosterveld, B.; Frasca, T.; and Platt, R. 2017. Spoken Instruction-Based One-Shot Object

and Action Learning in a Cognitive Robotic Architecture. In *Proceedings of the Sixteenth International Conference on Autoomous Agents and Multiagent Systems*, 1378–1386. São Paulo, Brazil: ACM.

Scheutz, M.; Schermerhorn, P.; Kramer, J.; and Anderson, D. 2007. First Steps toward Natural Human-Like HRI. *Autonomous Robots* 22(4): 411–423.

Scheutz, M.; Williams, T.; Krause, E.; Oosterveld, B.; Sarathy, V.; and Frasca, T. 2019. An overview of the distributed integrated cognition affect and reflection diarc architecture. In Ferreira, M. I. A.; S.Sequeira, J.; and Ventura, R., eds., *Cognitive Architectures*, 165–193. Intelligent Systems, Control and Automation: Science and Engineering book series (ISCA, Series) Springer.

Suddrey, G.; Lehnert, C.; Eich, M.; Maire, F.; and Roberts, J. 2017. Teaching robots generalizable hierarchical tasks through natural language instruction. *IEEE Robotics and Automation Letters* 2(1): 201–208.

Whitehead, S. D.; and Ballard, D. H. 1991. Learning to perceive and act by trial and error. *Machine Learning* 7(1): 45–83.