Cut & recombine: reuse of robot action components based on simple language instructions

Minija Tamosiunaite[1,2], Mohamad Javad Aein[1], Jan Matthias Braun[1], Tomas Kulvicius[1], Irena Markievicz[2], Jurgita Kapociute-Dzikiene[2], Rita Valteryte[2], Andrei Haidu[3], Dimitrios Chrysostomou[4], Barry Ridge[5], Tomas Krilavicius[2], Daiva Vitkute-Adzgauskiene[2], Michael Beetz[3], Ole Madsen[4], Ales Ude[5], Norbert Krüger[6] and Florentin Wörgötter[1]

Abstract
Human beings can generalize from one action to similar ones. Robots cannot do this and progress concerning information transfer between robotic actions is slow. We have designed a system that performs action generalization for manipulation actions in different scenarios. It relies on an action representation for which we perform code-snippet replacement, combining information from different actions to form new ones. The system interprets human instructions via a parser using simplified language. It uses action and object names to index action data tables (ADTs), where execution-relevant information is stored. We have created an ADT database from three different sources (KUKA LWR, UR5, and simulation) and show how a new ADT is generated by cutting and recombining data from existing ADTs. To achieve this, a small set of action templates is used. After parsing a new instruction, index-based searching finds similar ADTs in the database. Then the action template of the new action is matched against the information in the similar ADTs. Code snippets are extracted and ranked according to matching quality. The new ADT is created by concatenating code snippets from best matches. For execution, only coordinate transforms are needed to account for the poses of the objects in the new scene. The system was evaluated, without additional error correction, using 45 unknown objects in 81 new action executions, with 80% success. We then extended the method including more detailed shape information, which further reduced errors. This demonstrates that cut & recombine is a viable approach for action generalization in service robotic applications.

Keywords
Cognitive robotics, manipulation, manipulation planning, control architectures and programming, service robotics

1. Introduction

Programming of robots remains a tedious process, where trajectories as well as force and torque profiles must be determined and conveyed to the machine and grasp type and force must be determined, if necessary. In industrial applications, waypoint-based programming (Macfarlane and Croft, 2003) or teleoperation (Moradi Dalvand and Nahavandi, 2014) are most frequently used, with some involvement of kinesthetic teaching of waypoints (Fischer et al., 2016; Gaspar et al., 2017; Schou et al., 2013). Conversely, service robotics widely considers (semi)autonomous methods. The most traditional are learning by demonstration (Billard et al., 2008; Dillmann, 2004) and reinforcement learning (Kober et al., 2013). Reactive

[1]Department for Computational Neuroscience, Inst. Physics-3, Georg-August-Universität Göttingen, Germany
[2]Faculty of Informatics, Vytautas Magnus University, Lithuania
[3]Institute for Artificial Intelligence, University of Bremen, Germany
[4]Department of Materials & Production, Robotics and Automation Group, Aalborg University, Denmark
[5]Department of Automatics, Biocybernetics, and Robotics, Jožef Stefan Institute, Slovenia
[6]Maersk Mc-Kinney Moeller Institut, South Denmark University, Denmark

Corresponding author:
Minija Tamosiunaite, Georg-August-Universität Göttingen, Department for Computational Neuroscience, Inst. Physics-3 Friedrich-Hund Platz 1, D-37077 Göttingen, Germany.
Email: minija.tamosiunaite@vdu.lt

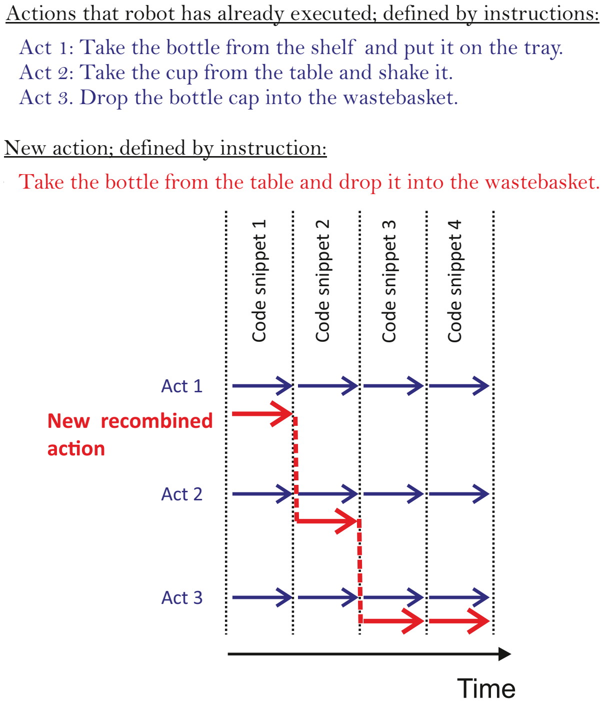
components, for example, for error correction, are often added here, too, to make the robotic system more robust (Erdem et al., 2015; Nakamura et al., 2013; Stulp et al., 2012).

The aforementioned industrially oriented methods require a lot of effort from specialists (programmers and system integrators), while learning methods remain far from autonomous and only groups with expertise in learning are able to develop working examples. Recently, research has also targeted the reduction of robot programming and training efforts. Here, usage of advanced visual interfaces (Huang et al., 2016; Schlette et al., 2014), also paired with touch or gestures (Profanter et al., 2015), natural language instruction (Bollini et al., 2013; Misra et al., 2016; Stenmark and Nugues, 2013; Tellex et al., 2011), knowledge-based methods (Beetz et al., 2016; Tenorth and Beetz, 2013), and advanced grasp and motion planning (Alterovitz et al., 2016; Bohg et al., 2014), allow the robot to behave in new environments.

We propose a framework for robot experience reuse, based on a recombinable data structure for actions. The structure allows code snippets to be cut from several existing action instantiations and put back together to represent a new action instantiation (see Figure 1 for a schematic representation). After validating the new action on a robot, we store this action instantiation in the database for future recombination and reuse. Thus, this approach might, over time, become very powerful, by making use of the fact that the database will continue to grow, allowing for more and more possible recombinations.

Of specific interest for us was the development of a system within a given larger application domain, essentially independent of the robot. To this end, the data structure introduced next allows for the storage of data from different sources (e.g., from a KUKA LWR, or UR5, or from a simulation), such that it is still possible to recombine the data from these different sources into a new execution protocol. We chose table-top manipulation actions as an application domain. This includes tasks in a kitchen but also small-part industrial assembly and chemical laboratory experimentation tasks. Hence, one goal of this study is to show that the cut & recombine method works across different tasks and different data sources.

As mentioned, to achieve this, the definition of an appropriate data structure and of the action recombination procedures are the core of the problem. On top of this, one needs to define a procedure that “tells the robot what to do”, without which the system would not know what to look for in the database to begin with. The latter, we address by using language-based instructions that can be understood by a human operator, such as: “Place the bottle on the shelf”, performing a parsing procedure that specifically links to the action instantiation database. To reduce the language-analysis effort, we constrain the instruction language to some degree, specifically requiring instructions to be phrased with an appropriate level of granularity. Language processing is not central to our study and, as a consequence, we differentiate ourselves from that group of existing systems that emphasizes the translation of fully complex natural language, ubiquitous and incomplete, into robotic execution (albeit usually in rather limited domains) (Bollini et al., 2013; Lisca et al., 2015; Misra et al., 2016; Tellex et al., 2011). We use simpler language than is used in these studies, more related to the way one would give an instruction to a child or a “newbie” in a workshop. This makes our approach quite intuitive and also accessible to new and non-expert users. It also leads to more robust language processing outputs and may result in a larger potential for penetrating different robotic applications.


Actions that robot has already executed; defined by instructions:

Act 1: Take the bottle from the shelf and put it on the tray.
Act 2: Take the cup from the table and shake it.
Act 3. Drop the bottle cap into the wastebasket.

New action; defined by instruction:

Take the bottle from the table and drop it into the wastebasket.

Fig. 1. New action recombination using code snippets from previously executed actions.

In summary, this article has three main contributions. (1) Definition of a hierarchically organized data structure for robotic action representation, which facilitates recombination. (2) A set of algorithms that allows sub-symbolic data reuse from previous robot executions by recombining snippets from existing actions. (3) A language link that allows reuse based on simple language commands.

The rest of the paper is organized as follows: we start with an overview of the approach in Section 2. Then we describe the model assumptions on which the data structures are based in Section 3. Afterwards, we describe data structures (Section 4) and procedures (Section 5) in full detail. Then we provide results on instruction text processing, as well as on recombination and execution of several new instructions in Section 6. Finally, we evaluate our approach and compare it with the state of the art in the discussion (Section 7).

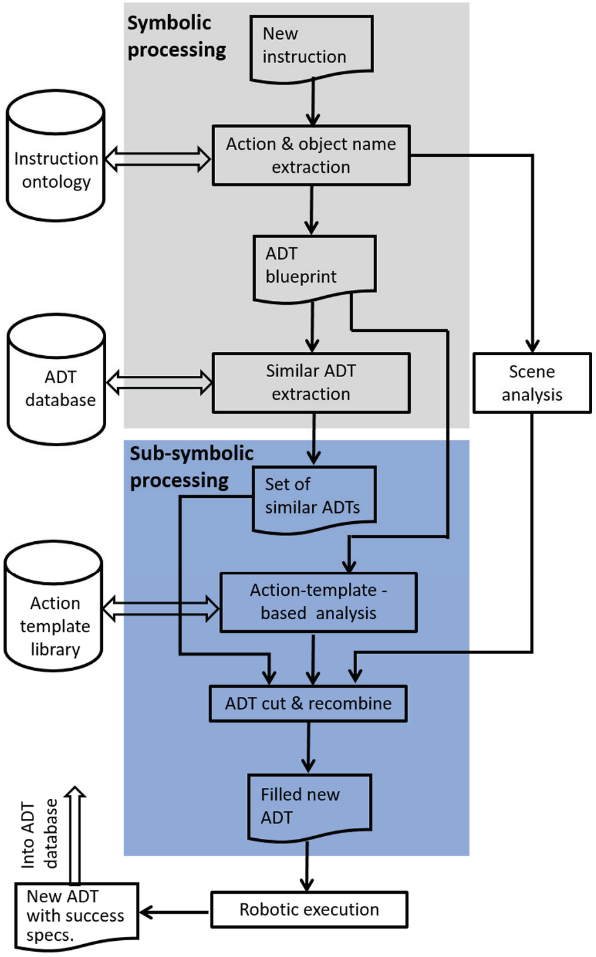


Fig. 2. Overview of the approach.
ADT: action data table.

2. Overview of the approach

Our system consists of three data structures and two main procedures (see Figure 2).

Data structures are:

- *Instruction ontology*, containing verbs and nouns for actions and objects, introduced to handle synonymy as well as robotics-related instruction parsing issues.
- *ADT database*, where “ADT” stands for “action data table”. An ADT is an XML data structure containing information from one previous robot execution of an action down to control level parameters. While sufficient for execution, the ADT also preserves the symbolic link to the instruction ontology. ADTs have a strict temporal structure allowing not only reuse of the complete ADTs but also recombination of the ADT snippets into new executable ADTs. A visualization explaining the main aspects of the ADT is presented in Figure 3.
- *Action template library*, where so-called action templates for a set of actions are stored. An action template is an abstract encoding of an action, where the temporal action structure is encoded in a systematic way. Action templates are indexed (named) by the action word (verb). The action template, as such, provides the scaffold for the recombination processes. To create a new ADT, the (abstract) bits and pieces of the relevant action template will have to be filled in with snippets from existing ADTs. The action template library provides a list of all here-investigated robot-executable actions.

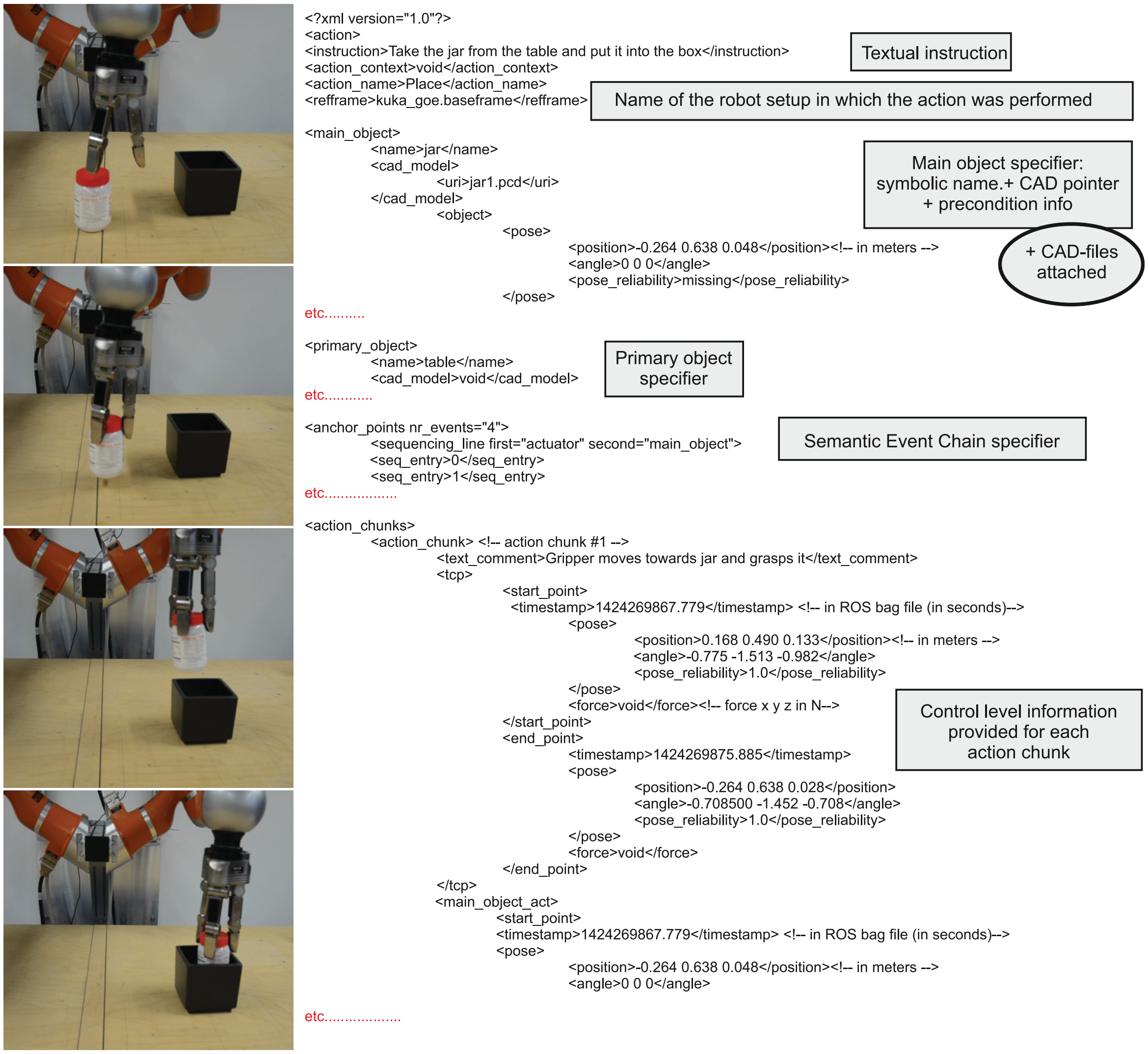
Thus, the goal of the system is to interpret a new instruction and create a *new* ADT, recombining snippets of ADTs stored in the ADT database.

The procedure consists of two main parts:

- *Symbolic processing* (Figure 2, top), where—given a new instruction—the corresponding action word (verb) and object names (nouns) are extracted. Object names are sorted according to the roles they play in the planned execution. Action and object names with object roles in the action are written into an empty ADT, creating the so-called ADT blueprint. Based on both action and object names in the new instruction, a set of similar ADTs is extracted from the *ADT database*.
- *Sub-symbolic processing* (a two-phased procedure, see the bottom part of the diagram in Figure 2), where the structural information from the action templates—the scaffold—is used to search for useful snippets in the set of similar ADTs. Those snippets are recombined to form a new ADT for the new instruction. For this, we also need to perform scene analysis in order to adapt control information to the poses of objects in the actual scene.

The triplet of instruction, action template, and ADT, together with their processing routines, can be viewed as components of a three-layer architecture. The instruction represents an action at purely symbolic level (top layer). The action template (middle layer) introduces an abstract temporal action structure, based purely on the action word in the instruction. Finally, the ADT (bottom layer) provides execution-level details for each temporal segment introduced in the action template. Note that the execution details stored in an ADT depend not only on the action as such, but also on the objects with which the action is performed, as well as on the object geometry and poses in the scene (all that information is provided in the ADT, as well). While symbolic processing takes place at the highest level, we employ two stages of sub-symbolic processing: action-template-based structural analysis of the new action (middle layer) and ADT-based snippet cutting and recombination (bottom layer).

Details of the data structures and the procedures are provided in Sections 4 and 5.



```
<?xml version="1.0"?>
<action>
<instruction>Take the jar from the table and put it into the box</instruction>
<action_context>void</action_context>
<action_name>Place</action_name>
<refframe>kuka_goe.baseframe</refframe>

<main_object>
        <name>jar</name>
        <cad_model>
                <uri>jar1.pcd</uri>
        </cad_model>
                <object>
                        <pose>
                                <position>-0.264 0.638 0.048</position><!-- in meters -->
                                <angle>0 0 0</angle>
                                <pose_reliability>missing</pose_reliability>
                        </pose>
etc..........

<primary_object>
        <name>table</name>
        <cad_model>void</cad_model>
etc...........

<anchor_points nr_events="4">
        <sequencing_line first="actuator" second="main_object">
        <seq_entry>0</seq_entry>
        <seq_entry>1</seq_entry>
etc.................

<action_chunks>
        <action_chunk> <!-- action chunk #1 -->
                <text_comment>Gripper moves towards jar and grasps it</text_comment>
                <tcp>
                        <start_point>
                         <timestamp>1424269867.779</timestamp> <!-- in ROS bag file (in seconds)-->
                                <pose>
                                        <position>0.168 0.490 0.133</position><!-- in meters -->
                                        <angle>-0.775 -1.513 -0.982</angle>
                                        <pose_reliability>1.0</pose_reliability>
                                </pose>
                                <force>void</force><!-- force x y z in N-->
                        </start_point>
                        <end_point>
                                <timestamp>1424269875.885</timestamp>
                                <pose>
                                        <position>-0.264 0.638 0.028</position>
                                        <angle>-0.708500 -1.452 -0.708</angle>
                                        <pose_reliability>1.0</pose_reliability>
                                </pose>
                                <force>void</force>
                        </end_point>
                </tcp>
                <main_object_act>
                       <start_point>
                       <timestamp>1424269867.779</timestamp> <!-- in ROS bag file (in seconds)-->
                       <pose>
                                <position>-0.264 0.638 0.048</position><!-- in meters -->
                                <angle>0 0 0</angle>
etc...................
```

Textual instruction
Name of the robot setup in which the action was performed
Main object specifier: symbolic name.+ CAD pointer + precondition info
+ CAD-files attached
Primary object specifier
Semantic Event Chain specifier
Control level information provided for each action chunk

Fig. 3. Main aspects of the action data table. Images on the left are provided only for visualization purposes and are not part of the action data table.

3. Model assumptions

We will first introduce the action model we are using in this study. Data structures will then follow from that model. The model encompasses elements from symbolic and sub-symbolic (control) domains and helps to close the gap between the human-understandable symbolic domain and the robot-executable control domain.

3.1. Action temporal structure

We perform temporal action chunking at two different hierarchical levels: semantic event chain level and movement primitive level (see Figure 4 for visualization of those levels).

- *Semantic event chain (SEC)*. This gives a symbolic definition of actions by encoding the sequence of touching and un-touching events between object pairs (Aksoy et al., 2011, 2017). This creates a well-defined and reproducible temporal chunking of actions. A chunk is a segment between two SEC (touching or un-touching) events.
- *Movement primitives*. We further divide each chunk into a sequence of movement primitives on the basis of trajectory segmentation (Aein, 2016). Each movement primitive corresponds to an elementary movement of the robot arm or gripper, such as moving to a goal position or grasping an object. The movement primitive list is discussed in detail in Subsection 4.2. Within each chunk, the given sequence of movement primitives should be executed to achieve the event related to the chunk.

3.2. Object roles

To make the action model independent of specific objects, we define objects based on the roles that they play in the action. These roles are determined by the types of change

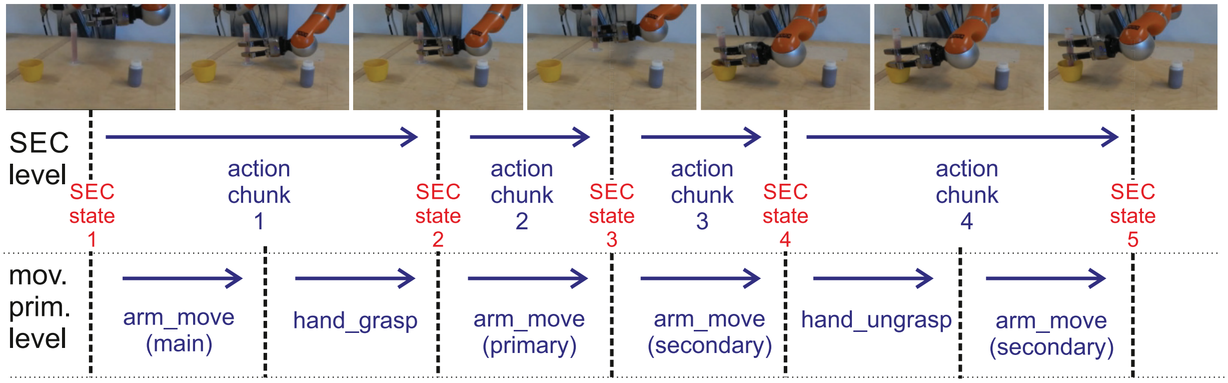


Fig. 4. Temporal action structuring at two different hierarchical levels: semantic event chain (SEC) and movement primitives (mov. prim.). Video frames are taken for the instruction "Place the measuring beaker into a pot." The SEC states (one to five) for this action are specified in more detail in Table 2. The full movement primitive sequence (here it was truncated at the ends for visualization purposes) is also given in the same table. The so-called object denominators are shown in parentheses under the movement primitives; this is explained in Subsection 4.2.

in object relations during the manipulation. An action starts and ends with the manipulator not touching or holding anything. From this, we get the following roles in our model:

1. *Manipulator*. The object that performs the action, for example a human or robot hand.
2. *Main*. The object that interacts directly with the manipulator.
3. *Primary*. An object that interacts with the main object. The relation of main and primary object changes from touching (T) to not touching (N).
4. *Secondary*. An object that interacts with the main object. The relation of main and secondary object changes from not touching (N) to touching (T).

In addition, we introduce supports: main support, primary support, and secondary support, for the main, primary, and secondary objects, respectively. At the start, the relations of objects and their corresponding supports are touching (T). In every action, we have at least the manipulator and the main object. The existence of other object roles depends on the action. For example, in the action defined by the instruction "Push the bottle away from the tray", the main object is the bottle and the primary object is the tray, while in the instruction "Pick the bottle from the shelf and place it on the tray", three object roles have to be defined: the main object is the bottle, the primary object the shelf, and the secondary object the tray.

3.3. Action granularity

As mentioned, in our framework, the start and end of an action are clearly defined: an action starts and ends with a free manipulator, which means that the manipulator does not touch any other object. Between these two states, the manipulator approaches the main object, touches it, and performs the action. The reasoning behind defining atomic actions in this way is discussed at great length by Wörgötter et al. (2013). One advantage is that in this way we can divide a long demonstration into smaller meaningful actions in a reproducible way. We can also execute a long task by sequencing several smaller actions. In addition, such an action definition enables well-defined instruction-to-action mappings to be made, as described next.

Table 1. List of action templates (given by action names).

Align	Lay	Pull	Put over
Chop	Place (top-top)	Punch	Rotate
Cut	Place (top-side)	Push	Screw
Drop	Place (side-side)	Push apart	Shake
Insert	Poke	Push to	Stir
Invert	Pour	Push from to	Unscrew

3.4. Language link

We execute instructions, which are formulated using so-called "robotic action words". These are action words that describe actions for which action templates exist in the action template library and, thus, are robot-executable in our system. The list of these actions is provided in Table 1. We also define as robotic action words the verbs defining parts of the action, such as *pick up, fetch*, and *grasp*. The central requirement for an instruction is that only action words from the robotic action list (or synonyms) are used, e.g., "Pick up the bottle and place it on the tray", "Shake the bottle", and "Shake the bottle and place it on the tray" would all be valid instructions within our requirements. We do not compile instructions if they are given using action words for which the property "robotic" is false (outside the list). E.g. "Throw away the empty bottle" has an action word *throw away* that is not in the "robotic action list" and thus would not be compiled. Such an instruction, alternatively, can be expressed using robotic action words, e.g. "Drop the empty bottle into the wastebasket"; if the wastebasket has a lid, the task can be extended into a sequence of instructions: "Put the wastebasket lid on the table","Drop the empty bottle into the wastebasket", and "Put the wastebasket lid on the wastebasket".

As shown in these examples, we allow more than one robotic action word to be mentioned in the instruction. To resolve this ambiguity, we define the action word property "central". This is the action word based on which the action template is chosen for execution. Thus, this action word must not be omitted in the instruction. In the examples containing two action words, "Pick up the bottle and place it on the tray" and "Shake the bottle and place it on the tray", the central action words are *place* (first instruction) and *shake* (second instruction), respectively. The remaining action words in the instruction, we call "supportive". The action words that do not have separate action templates (*pick* or *grasp*) are always supportive, while the action word *place* plays the role of the central action word in the first instruction but the role of the supportive action word in the second instruction.

ADTs are only labeled with respect to the central action word in the instruction. The central and supportive action words are distinguished in the instruction parsing procedure, as described in Section 5.1.

4. Data structures

Here, we provide a detailed explanation of the three main data structures introduced in Figure 2, adhering to the action model described in the previous section. We also briefly discuss how the databases were initially filled.

4.1. Data structure 1: instruction ontology

To form the *instruction ontology*, we use WordNet (Miller et al., 1990) subsets separately for action words and object names. In this study, we are mainly interested in WordNet synsets, that is, groups of synonym words, which allow us to resolve synonymy in the instructions (e.g., we want action words *put* and *place* in the instructions to be treated as the same word). The WordNet subset for action words was formed manually (by choosing robotic-action-compatible verb senses), based on the action template names existing in the *action template library* (see Table 1) and expanded by action names extracted from a set of sample instructions. Sample instructions were obtained using video transcriptions, either readily provided on the Internet (eleven videos) or transcribed by a small group of human participants (four videos transcribed by three participants). The video transcripts were needed to discover frequently used alternative formulations for descriptions of actions from the action template library, e.g. the action word *insert into* in a robotic sense is a synonym of *insert*, but this relation is not provided in WordNet. We used a combination of videos from robotic assembly of small parts and chemical laboratory operations. By using such a combination, we could cover the range of most frequent everyday actions (e.g., *place, insert*, and *turn* were shown in the industrial assembly videos, while *pour, shake, screw, unscrew*, and *invert* were typical for the chemical experiment videos). We added the earlier described binary-valued action properties "robotic", "central", and "supportive" to the action words in the ontology. Finally, the ontology was fine-tuned using the a sample of 250 instructions from a set of 500 instructions that we had created for evaluating our procedures in this study. The instructions were created by a group of three people who knew the language limitations for instructing the robot but did not have knowledge of the inner workings of the symbolic processing employed in this study.

For the object ontology, object names were taken from the sample instruction sets. Here again, the appropriate senses of nouns were chosen and WordNet subsets corresponding to those senses were extracted. All in all, we were working with an ontology having 67 action classes (113 action names, when considering synonyms) and 305 object classes. While these numbers seem small, it should be noted that for manipulation on a table top in the kitchen, chemical laboratory, or small industrial assembly not many more actions exist. Object classes can be easily extended to give many more actions; however, these were not yet needed for our experiments.

Action and object names in the ontology were linked to the ADTs. We organized this link by providing metadata for the ADTs contained in the database. Metadata introduce the relation between the ADT file name and the following set of names: central action and main, primary, and secondary objects in the ADT. This allows tracking back of which ADTs are associated to a given action or object name appearing in the *instruction ontology*.

4.2. Data structure 2: action templates

Action templates are abstract action encodings following the action model described in Section 3. One action template represents one action (and its synonyms). Action templates are based on the action library developed by Aein et al. (2013). In our work, we used 24 action templates from the manipulation action ontology presented by Wörgötter et al. (2013) (see Table 1). Note, as discussed next, that these action templates form rigorous scaffolds for the different actions, to allow allocation (and recombination of) snippets.

In an action template, we provide a sequence of SEC-based action chunks and a sequence of movement primitives in each SEC-defined chunk, based on abstract object roles (main, primary, secondary, etc.). An example of an action template for the action *place* is given in Table 2 (note that we label actions according to the central action word; thus, for consistency, we will be using action name *place*

Table 2. Action template in tabular form for action *place*. The semantic event chain (SEC) is given in the top part, with five states (Roman numerals) and touching (T) and non-touching (N) relations for different object pairs in these states. Below, it is indicated that four action chunks (Arabic numerals) are formed as transitions between the five SEC states. Movement primitive sequences belonging to each action chunk are indicated in the bottom line; each movement primitive is denoted P_{ij}, where i is the action chunk number and j is the number of the movement primitive in that action chunk. Concrete movement primitives, in the format *name(object denominator)*, are shown below the table.

	SEC					
	States	(I)	(II)	(III)	(IV)	(V)
Relations	hand, main	N	T	T	T	N
	main, primary	T	T	N	N	N
	main, secondary	N	N	N	T	T
	main, p.s.	N	N	N	N	N
	main, s.s.	N	N	N	N	N
	Action chunks	1	2	3	4	
	Movement primitives	P_{11} P_{12} P_{13}	P_{21}	P_{31}	P_{41} P_{42} P_{43}	

hand_pre(main) arm_move(primary) arm_move(sec.) hand_ungrasp
arm_move(main) arm_move(sec.)
hand_grasp arm_move(free)

p.s.: primary support; sec.: secondary; s.s.: secondary support.

instead of the more frequently used *pick & place*). Let us explain the notation in the table in detail.

In the upper part of the table, the SEC information is provided: that is, information of touching (T) and untouching (or non-touching, N) of object pairs throughout the action. The leftmost column shows the object pairs for which the SEC relations are calculated. Objects are given in an abstract way, according to their roles. All other columns show a single SEC state each, where the transitions between two SEC states are the action chunks.

Beneath each SEC column in the table, we show the sequence of movement primitives required to perform the action chunk. We indicate the sequence of movement primitives by labels (P_{11}, P_{12}, P_{13}, etc.), and below we specify the movement primitive name and the *object denominator* indicated in the brackets.

In the action template, we only consider movement primitives at the symbolic level (i.e., only movement primitive names are given, where the movement primitive set that we used is indicated in Table 3). For real execution, all movement primitives must have control level parameters, as indicated in the second column in Table 3. These control level details are *not* indicated in the action template. Note that the movement primitive list we are using is quite standard, as arm-hand systems often use a similar movement primitive list (Aksoy et al., 2016; Manschitz et al., 2014; Stenmark et al., 2015).

Table 3. Movement primitives with parameters. The third column indicates which parameters are extracted from action data tables (ADTs).

Movement primitive	Parameters	Parameter source
arm_ move	TCP pose Main object pose Primary object pose Secondary object pose Start time End time	ADTs
arm_rotate	Rotation axis Rotation angle Start time End time	Default
arm_move_periodic	Frequency Amplitude in X Amplitude in Y Amplitude in Z Start time End time	Default
hand_pre	Opening width	ADTs
hand_ungrasp	No parameters	
hand_grasp	Gripping force	Default

TCP: tool center point.

The object denominators (*main, primary, secondary*, or *free*, given in Table 2 in parentheses) are provided for movement primitives *arm_move* and *hand_pre*, where the latter is the pre-shaping of the hand. Object denominators specify which objects are to be dealt with by a certain movement primitive and are used to enable linking to the actual objects, as given in existing ADTs. Thus, for example, in the action template, the object denominator *main* provides information that the robot arm movement has to be interpreted with respect to the main (and not any other) object.

The relational meanings of the object denominators *primary* and *secondary* are given in Table 4. We also use the object denominator *free*, which specifies that the movement

Table 4. Relations expressed by object denominators.

Object denominator	ADT information to be reused
main in *arm_move*	Relation between TCP and main object
primary in *arm_move*	Relation between main and primary object
secondary in *arm_move*	Relation between main and secondary object
free in *arm_move*	Movement is object-independent
main in *hand_pre*	Pre-grasp width, defined by main object

ADT: action data table; TCP: tool center point.

primitive is independent of objects in the scene. In the context of the movement primitive *hand_pre*, we used an object denominator to declare pre-grasp width; see the last line in Table 4. Some movement primitives in our setting (e.g., *hand_grasp* and *hand_ungrasp*) are parameter-free and thus require no object denominators.

4.3. *Data Structure 3: Action Data Tables (ADTs)*

The ADT is a data structure that provides control level information as well as the symbolic-to-control link. An ADT consists of a header and body and is coded in XML. In the ADT header, the following items are provided:

- initial language instruction;
- central action name;
- main, primary, and secondary object names;
- object dimensions and weight (when available);
- links to object 3D models (when available);
- precondition as poses of main, primary, and secondary objects;
- SEC of the action;
- name of the robot or simulation setup in which the action is performed.

In the ADT body, action chunk and movement primitive information is provided at control level. The ADT body is structured on the basis of action templates and keeps the following information for each action chunk:

- start time;
- end time;
- TCP start pose;
- TCP end pose;
- main, primary, and secondary object start poses;
- main, primary, and secondary object end poses;
- a sequence of movement primitives with parameters as described in Table 3;
- grasp information (if grasp is present) in an action chunk;
- success specifier.

All information in the ADT is given in absolute coordinates. Thus, ADT information can only be reused directly in the same setup. To adapt to different setups, relative information between different entities represented in the ADT must be extracted. This can be achieved via coordinate transforms.

4.3.1. *Initial filling of the ADT database*

The ADT database grows through the cut & recombine approach but we had to kick-start it. Thus, the basis for our experiments was a database of 28 ADTs for 10 different actions performed using different objects. This ADT list is found in Section 6, needed there to better understand our final observations (see Table 10 in Section 6).

It is important in the cut & recombine method that ADT information should transfer across similar robotic systems. Hence, eight of those ADTs were acquired using the KUKA LWR arm with Schunk SDH2 gripper, the same as used in the test experiments; three ADTs were acquired using a Universal Robot Arm UR5 with Schunk WSG50 gripper (Kramberger et al., 2016); and the remaining 17 ADTs were made in simulations using a Razor Hydra device and the robotic simulator *Gazebo*, as described by Haidu and Beetz (2016).

All these ADTs were created using different conventional robot programming and simulation methods; the data were semi-automatically extracted and stored as described briefly in the following.

To extract ADTs from robot programs, action and object names (ADT header) were entered manually. Semantic event chains (Aksoy et al., 2011, 2017) were extracted based on video information (augmented by touch sensor readings); in this way, action chunks were obtained. Within these chunks, arm and gripper movement segmentation was performed as described by Aein (2016), where the standard approach of velocity change (Buchin et al., 2011; Kong and Ranganath, 2008) was employed for segmentation. In addition, an ADT editor tool suite was developed and employed to verify the obtained segmentation. This suite of tools consists of both a command-line tool and a graphical user interface (GUI) editor. The command-line tool generates new, or populates existing, ADT XML files using ROS bag recordings, either by making use of specialized binary topics in the ROS bag file, indicating how the bag file recordings should be parsed into ADT data chunks, or by taking such annotations as manual input arguments via intuitive point-and-click annotation along the action timeline.

To extract ADTs from Gazebo simulations, symbolic information was extracted and stored using the web ontology language OWL (for the ADT headers) and low-level data were saved into a MongoDB database. The tool suite, discussed previously, was extended by tools for transforming MongoDB knowledge entries into sub-symbolic data for the ADTs.

5. Procedures

In this section, we specify the algorithms we are using in symbolic and sub-symbolic processing, briefly introduced in Section 2.

5.1. Symbolic processing

The symbolic processing has two parts: (1) parsing the provided instruction for action and object name and role extraction and (2) finding similar existing ADTs according to the extracted action and object names.

Action and object name extraction is based on instruction syntactic analysis. Syntactic annotation is performed using the Stanford Parser (de Marneffe and Manning, 2008). Parsing errors are corrected using a dictionary of predefined syntactic roles, which are extracted from a reference set. Parsing errors occur because the Stanford Parser is not adapted to instruction parsing. Obtained dependency tree nodes are then analyzed by matching them with Semgrex patterns (Chambers et al., 2007): head-dependent relations are recognized using predefined regular expressions.

To parse a syntactic dependency tree, we use the modified Breadth First Search (BFS) algorithm, which includes static combinational logic blocks (Nivre and Nilsson, 2005). We assume that a parsed sentence is a directed acyclic graph of words. Each word, depending on its syntactic role, activates a set of logic rules, which are then used to process further tree nodes. The sequence of rule execution is important and proceeds down the rooted tree. First, we identify the central action, then the main object, and, finally, the primary and secondary objects. Our algorithm performs the following steps:

1. **Identify central action**. The dependency tree is a directed acyclic graph with the verb as *root*, where each word appears exactly once (Klein and Manning, 2004). When there is only one verb in the sentence, the root identifies the central action. If there are several verbs, the relations between the verbs are analyzed. For the relation *conj* expressed by conjunction *and* (e.g., in the instruction "Pick up the bottle and place it on the tray"), we query the instruction ontology to disentangle which verb denotes the central action (see Algorithm 1). For other conjunctions (e.g., *after, although*, or *because*), the root verb is considered to be the central action.
2. **Identify multiword expression defining the central action**. If the link between the central verb and some other word in a sentence describes phrasal, particle, or serial relations (dependency relations[1]: *compound: prt, compound :svc or aux*), the word is attached to the expression of the central action. For example, using the mentioned relations, the instruction "Put down the bottle" is parsed with the central action *put down. Multiword central action expressions* are recognized using finite or non-finite clause expressions (dependency relations: *ccomp*, *xcomp*). In the example "Start mixing the liquid" the word *mixing* is identified as the clausal complement of the verb *start* and thus serves as the central action.

Algorithm 1 Procedure for choosing the central action.

Inputs:
- A list of action words that have relation *conj* in the dependency tree obtained from the instruction.
- Instruction ontology indicating properties *central_action* and *supportive_action* for action words.

Output:
- The action word for central action in the instruction.

```
procedure CENTRAL_ACTION
    candidate_list = ∅.
    Move all action words connected by relation conj to the candidate_list.
    Delete action words for which property central_action = 0 from the candidate_list.
    if more than one action word is remaining in the candidate_list and there are action words for which property supportive_action = 0 then
        Delete action words for which property supportive_action = 1 from the candidate_list.
    if only one action word remains in the candidate_list then
        The action word denotes central action.
    else
        Human intervention is required.
```

3. **Identify main object**. The core argument of the root verb is the subject (dependency relation: *nsubj*) which is normally omitted in instruction sentences. The second dependency after the subject is the object. It is recognized with nominal arguments: *nsubjpass*, *dobj* (de Marneffe et al., 2014). For example, in a sentence "Place the *pot* on the table", the noun *pot* is identified as the direct object of the root verb *place*. In the robotic instruction, it takes the main object's semantic role. The use of passive forms of the subjects is handled in the same way: e.g. in the sentence "The pot shall be placed on the table", the noun pot is identified as the main object of the passive verb *placed*.
4. **Identify multiword expressions defining the main object**. To identify the noun or noun phrase and its relations, we use the *nmod* dependency. For example, "Pour the content of the bottle" is parsed with the main object *content of bottle*. We use a collocation list to distinguish adjectival modifiers (relation *amod*, e.g., "Get the red bottle": *amod*(*bottle, red*)) from collocation expressions, e.g., "measuring beaker" (collocations are word sequences that occur more often than would be expected by chance and have special meanings). The collocation list was prepared using a domain-specific corpus, by calculating the *logDice* coefficient (Markievicz et al., 2013).
5. **Identify primary and secondary objects**. Definition of the primary and secondary objects is based on the

Table 5. Action template in tabular form for action *drop* with the neighborhood of movement primitive P_{12} indicated in blue. Table reads as follows: in the top part, the semantic event chain (SEC) is given with five states (Roman numerals) and touching (T) and non-touching (N) relations shown for different object pairs in these states. Below, it is indicated that four action chunks (Arabic numerals) are formed as transitions between the five SEC states. In the bottom line, movement primitive sequences for each action chunk are indicated; each movement primitive is denoted P_{ij}, where i is the action chunk number and j is the number of the movement primitive in that action chunk. Concrete movement primitives in the format *name(object denominator)* are shown below the table.

	SEC					
	States	(I)	(II)	(III)	(IV)	(V)
Relations	hand, main	N	T	T	N	N
	main, primary	T	T	N	N	N
	main, secondary	N	N	N	N	T
	main, p.s.	N	N	N	N	N
	main, s.s.	N	N	N	N	N
	Action chunks	1	2	3	4	
	Movement primitives	P_{11} P_{12} P_{13}	P_{21}	P_{31} P_{32}	P_{41}	

hand_pre(main) arm_move(prim.) arm_move(sec.) arm_move(free)
arm_move(main) hand_ungrasp
hand_grasp

prim.: primary; p.s.: primary support; sec.: secondary; s.s.: secondary support.

nmod dependency relation among indirect connections with respect to the central action. The relation *nmod* is used with different types of prepositions: place prepositions (e.g., in, on, at), direction prepositions (e.g., to, toward, through, into) and device prepositions (e.g., by). The definitions of primary and secondary objects are based on preposition types and the thematic role of the action verb from VerbNet (Kipper et al., 2006). We read each verb–preposition pair and compare it with the pre-built VerbNet frame lists, separately for primary and secondary objects. For example, the verb *place* in VerbNet has the thematic role *destination* and the lexical frame *NP V PP.destination NP.* Encompassing this thematic role allows the secondary object to be recognized.

After extracting action and object names, we record them in the otherwise empty ADT, in this way producing an ADT blueprint. In addition, based on the extracted names, a set of ADTs is extracted from the database, where at least one of the symbolic names matches. These are candidate ADTs for extracting control information in the sub-symbolic processing phase.

5.2. Sub-symbolic processing

Here, we recombine information from existing ADTs into a new ADT for a new instruction. Two stages of processing are used:

1. Abstract action-template-based analysis;
2. Cutting snippets from existing ADTs and recombining them into a new ADT.

The action template usage in the algorithm is twofold. First, an appropriate action template is used to extract the movement primitive sequence required for execution of the new instruction. Second, abstract movement primitive replacement lists are formed based on action templates. Searching for concrete control details (snippets in the existing ADTs) is then based on those lists.

Here, we show by an example what is meant by movement primitive sequence extraction and then proceed to a detailed description of the action-template-based analysis. For example, for the instruction "Drop the bottle into the wastebasket", we would use the action template for the action *drop* (Table 5), where the following movement primitive sequence is given: *hand_pre(main), arm_move(main), hand_grasp, arm_move(prim.), arm_move(sec.), hand_ungrasp, arm_move(free)*. Object denominators are shown in the parentheses. The movement primitives without object denominators (here, *hand_grasp* and *hand_ungrasp*) are parameter-free, thus, no information from previous execution is needed. The movement primitives with object denominators (all others) require snippet extraction from the existing ADTs; a detailed explanation of this procedure is given next.

5.2.1. Action-template-based analysis. This analysis is based on the similarity of so-called *neighborhoods of movement primitives* within different actions. Specifically, we consider the self-inclusive temporal neighborhood, both at the level of the movement primitive sequence and at the higher hierarchical level of semantic event chain states.[2] An example of the neighborhood of a movement primitive P_{12} is given in Table 5 using blue font. The exact procedure of the neighborhood definition is given in the appendix.

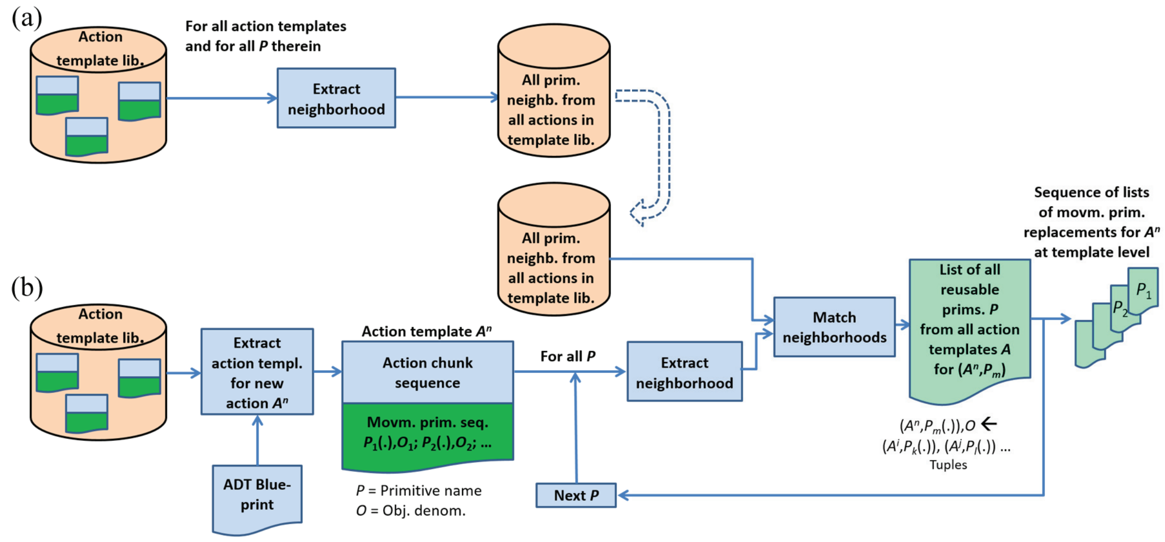

(a)
Action template lib.
For all action templates and for all P therein
Extract neighborhood
All prim. neighb. from all actions in template lib.
(b)
Action template lib.
Extract action templ. for new action A^n
ADT Blue-print
Action template A^n
Action chunk sequence
Movm. prim. seq. $P_1(.),O_1; P_2(.),O_2; ...$
P = Primitive name
O = Obj. denom.
For all P
Next P
Extract neighborhood
All prim. neighb. from all actions in template lib.
Match neighborhoods
List of all reusable prims. P from all action templates A for (A^n,P_m)
$(A^n,P_m(.)),O$ ←
$(A^i,P_k(.)), (A^j,P_l(.))$ ...
Tuples
Sequence of lists of movm. prim. replacements for A^n at template level
P_1
P_2

Fig. 5. Action-template-based replacement list formation. (a) Extraction of movement primitive neighborhoods from all action templates. (b) Movement primitive replacement list formation procedure. The inputs are the ADT blueprint, the action template library, and the set of all movement primitive neighborhoods extracted in part (a). The output is the sequence of lists of movement primitive replacements indicated on the right. The notation $P_1(.)$, $P_2(.)$, $P_m(.)$ means symbolic movement primitive names without concrete parameters. Movement primitives here are labeled by a single index (as opposed to the double-index used elsewhere in the paper) to simplify the notation. A^n labels the new action. The object denominator O for A^n is saved together with the replacement list.

We assume that a movement primitive of one action can be replaced by the movement primitive of the same or a *different* action where the neighborhoods of the movement primitives match. Let us show by an example that reuse of movement primitives from a different action is also viable. Let us assume that we have an ADT for the instruction "Place the bottle on the shelf" (the action template for *place* is provided in Table 2) and that the new instruction is "Drop the bottle into the wastebasket" (the action template in Table 5). One can observe that the emphasized neighborhood of movement primitive *arm_move(main)* for the action *drop* (Table 5) corresponds to the neighborhood of the analogous movement primitive *arm_move(main)* in the action template for the action *place*. Thus, we include the movement primitive *arm_move(main)* from action *place* in the replacement list of the movement primitive *arm_move(main)* for the action *drop*. This corresponds to human judgment that one can most probably approach the bottle with the arm for dropping it the same way as the bottle has been approached for the *place* action.

Now we will proceed to the algorithmic details of formation of the movement primitive list for potential use in a new ADT. The algorithmic procedure is shown in Figure 5. The procedure is as follows:

- First, we extract a set of all possible movement primitive neighborhoods from the action template library (Figure 5(a)).
- Then we extract the action template indicated in the ADT blueprint by the central action name and extract movement primitives in a sequence from that template (Figure 5(b)).
- For each of the movement primitives in the action template for the new action, we extract the neighborhood.
- Finally, we search the entire extracted set of neighborhoods for matches with the neighborhood of the new action movement primitive (right side of Figure 5).

Table 6. Replacement list for movement primitive (1,2), action *drop*. Pairs of indexes denote the number of the action chunk and the number of movement primitive in the action chunk in the action template.

What to replace		With what to replace	
Action	Index	Action	Index
Drop	(1,2)	Drop	(1,2)
		Insert	(1,2)
		Lay	(1,2)
		Place	(1,2)
		PutOver	(1,2)
		Screw	(1,2)
		Shake	(1,2)
		Unscrew	(1,2)

In this way, we make a list of possible replacements for *each* movement primitive of the new action. An example of the result of this procedure is given in Table 6, where the replacement list for the movement primitive *drop* $(1, 2)$ is shown. Pairs of indexes indicate: (number of the action chunk, number of movement primitive in the action chunk).

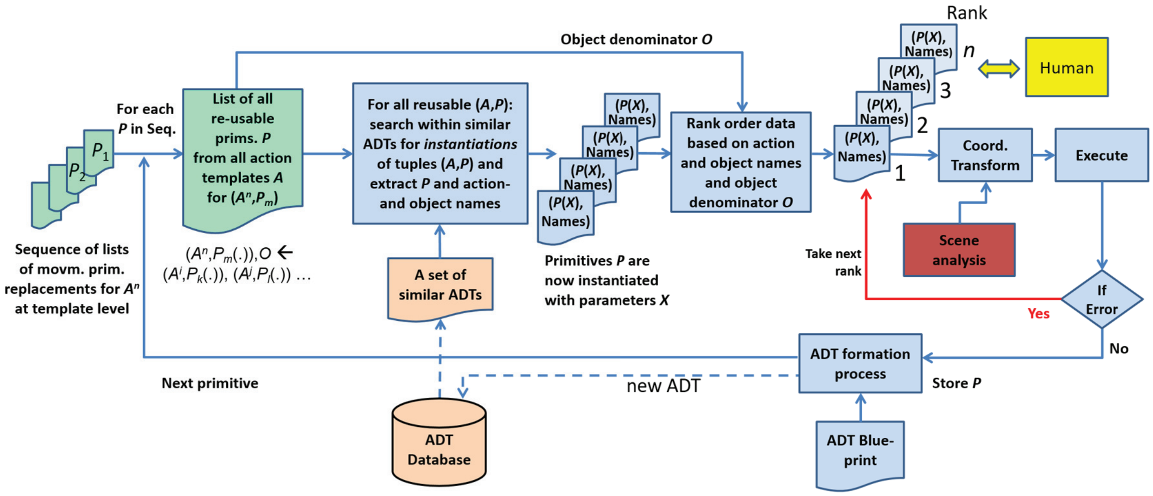

Object denominator O
Rank
(P(X), Names) n
(P(X), Names) 3
(P(X), Names) 2
(P(X), Names) 1
Human
For each P in Seq.
P1
P2
List of all re-usable prims. P from all action templates A for (A^n,P_m)
For all reusable (A,P): search within similar ADTs for instantiations of tuples (A,P) and extract P and action- and object names
(P(X), Names)
Rank order data based on action and object names and object denominator O
Coord. Transform
Execute
Scene analysis
Sequence of lists of movm. prim. replacements for A^n at template level
(A^n,P_m(.)),O ← (A^i,P_k(.)), (A^j,P_l(.)) …
A set of similar ADTs
Primitives P are now instantiated with parameters X
Take next rank
Yes
If Error
No
ADT formation process
Store P
Next primitive
new ADT
ADT Database
ADT Blue-print

Fig. 6. Cutting and recombining snippets of action data tables (ADTs) based on replacement lists. Inputs are replacement lists (on the right) and a set of similar ADTs, as well as ADT blueprints formed in the symbolic processing stage. The output is robotic execution of the new instruction and the finished ADT for the performed execution. The notation $P(X)$ means movement primitive instantiated with control parameters. Other notation comes from Figure 5.

Table 7. Rank orders for movement primitive replacement with different object denominators, showing which symbolic items have to match in order to achieve the rank, for three different cases.

Rank	Case 1 Object denominator *main*	Case 2 Object denominator *primary*	Case 3 Object denominator *secondary*
1	all	all	all
2	act. + main + sec.	act. + main + prim.	act. + main + sec.
3	act. + main + prim.	act. + prim. + sec	act. + prim. + sec.
4	act. + main	act. + prim	act. + sec.
5	main + prim. + sec.	main + prim. + sec.	main + prim. + sec.
6	main + prim.	main + prim.	main + sec.
7	main + sec.	act. + main	act. + main
8	main	act + sec.	act. + prim.

act: central action name; main: main object name; prim.: primary object name; sec.: secondary object name

Clearly, the movement primitive can be replaced by the same movement primitive from the same action *drop*, but it can also be replaced by movement primitives from actions *insert, lay, place*, etc. We make such replacement lists for all movement primitives requiring replacements in the new action, as shown on the right side of Figure 5.

5.2.2. Cutting and recombining snippets from ADTs. In this step, we cut appropriate snippets with control parameters from existing ADTs and recombine them to obtain an executable ADT for the new action. A snippet in our formalism essentially corresponds to a parametrized movement primitive. We search for snippets in the ADTs based on the replacement lists made in the action-template-based analysis step.

While we only considered action names in the action-template-based analysis, here we also take object names into account. We make the assumption that for movement primitives from the same replacement list performed with similar objects, the movement will be similar. Note that as we are talking about generalization here, we only require that this assumption holds in most cases; we do not expect to achieve full 100% performance.

The algorithm is specified in Figure 6. The input to the algorithm is the sequence of replacement lists (see output from the previous algorithmic procedure, Figure 5, right side). We analyze one list at a time. For each possible replacement of a movement primitive in the list, we search for instantiations in a set of similar ADTs. We cut out the discovered instantiations of these movement primitives from the ADTs and save them, together with symbolic action and object names (also obtained from ADTs). In this way, we obtain a set of different ADT snippets: candidates for replacement of one movement primitive in the new instruction. We use symbolic names to rank the extracted snippets. The ranking rules are provided in Table 7. We use different ranking rules, given different object denominators. The reasoning behind this is the following: if one performs a movement with respect to some role of objects (e.g., main, primary, or secondary), the corresponding object becomes more important in the ranking. Otherwise (when comparing objects that are not indicated in the object denominator), we consider the main object more important than primary and secondary objects.

In addition to symbolic-name-based ranking, we have implemented a hybrid ranking procedure, taking both symbolic and sub-symbolic similarity of ADTs into consideration. To evaluate the sub-symbolic similarity, we have compared the bounding boxes (in a real scene compared with in an ADT) of the object given in the movement primitive denominator (main, primary, or secondary). This allows object size and aspect ratio to be compared, where the latter is a shape-related parameter. To obtain the hybrid measure, we re-implemented the symbolic ranking given in Table 7 on the basis of a weighting procedure, thus obtaining the similarity value S_{symb} in the interval $[0, 1]$. To compare object bounding boxes, we use the intersection over union (IoU) measure to obtain another value S_{box} in the interval $[0, 1]$ (for details on both measures see the appendix). We define the hybrid similarity measure S_{h} by applying the weighted average of S_{symb} and S_{box}

$$S_{\text{h}} = \theta S_{\text{box}} + (1 - \theta) S_{\text{symb}} \tag{1}$$

where θ is the weight in the interval $[0, 1]$; we show results for the complete interval of θ values in the Section 6 (see Figure 8 in that section). We rank the snippets according to S_{h}.

From here on, one can now concatenate the (top-ranked) snippets for each movement primitive required in the execution of the new instruction and form the new ADT, as discussed next.

5.2.3. New ADT formation, execution, and storage. The previously described automatic procedure renders a rank list of the different snippets for recombination. However, because snippets come from foreign actions with different objects, fully automatic selection of snippets following their ranking will, in rare cases, lead to execution failures (e.g., when object sizes are too different), which would be detected only *after* robotic execution. To save time (and avoid looping through such unsuccessful executions), we have here built in one check by the user. If the user discovers, according to his or her expert knowledge, that a certain snippet will very probably not work, we allow the system to choose the next best from the rank list. This procedure is indicated in Figure 6 on the right side (yellow).

In addition, the actual visual scene configuration needs to be taken into account (Figure 6, red box). This involves extracting the object location and orientation. As this is a technical aspect, details are given in the appendix, where we also show how to perform coordinate transformation from the object coordinates given in the ADT to the actual scene coordinates.

After completion of recombination, the action will be executed and, in case of success, we insert the movement primitive with control parameters in the new ADT for further ADT storage in the database (bottom part of Figure 6).

This concludes all procedures. Several smaller additional algorithmic details are described in the appendix.

6. Results

6.1. Symbolic processing

We have used a set of 500 instructions of five different levels of complexity (100 instructions for each level) and analyzed them using the parser described in Section 5.1. The five complexity levels are:

(a) Simple instructions, where only one central robotic action word is present and object names are simple (e.g., "Invert the book");
(b) Instructions with several action words, where both central and supportive action words are present but object names are kept simple (e.g., "Take the book and invert it");
(c) Instructions where only the central action word is provided but objects have object identifiers (e.g., "Invert the second book");
(d) Instructions with both: several action words and objects with identifiers (e.g., "Take the story book and invert it");
(e) Instructions presented in passive form (e.g., "The second book must be inverted").

We used half of the instruction set (50 in each category) to tune the instruction ontology (as described in Section 4.1) and the symbolic processing procedure (as described in Section 5.1). The other half was used for testing. Test results are shown in Table 8.

Within the assumed reduced instruction language complexity, these results show that the symbolic processing procedure produces only isolated mistakes.

6.2. Sub-symbolic processing

We have investigated the cut & recombine approach by performing on a robot a test set of ten instructions that the robot had not executed before. The instructions are presented in the first column of Table 9. For execution we used a KUKA LWR robot arm with Schunk SDH2 gripper. First we used the symbolic-name-based snippet ranking procedure as described in Table 7 and further extended the study with the hybrid ranking procedure.

Note that the performed analysis is strictly feed-forward. Hence, no error correction mechanisms or reactive control policies were added, because we wanted to analyze how the cut & recombine approach performs on its own.

To make a comparison with a baseline method, we have performed a subset of these test instructions using an object-independent action library (Aein et al., 2013). This

Table 8. Error rate in instruction parsing into: central action, main, primary, and secondary objects. For each case, $n = 50$.

Instruction class	Error rate, %				
	Central action	Main object	Primary object	Secondary object	Instruction in general
Simple instructions	0	0	0	0	0
Several action words	0	2	2	0	4
Objects with identifiers	0	0	0	2	2
Several action words and objects with identifiers	0	2	0	0	2
Passive form	2	0	0	2	4

Table 9. Success rate of the recombined actions as well comparison to the success in case of using "object-independent" actions for 10 instructions. Where not indicated differently in the Remarks column, all the first hits in the ranked movement primitive lists were used. The same instruction was executed with three to ten different object–position combinations, as indicated by the number behind the slash in columns 3 and 4.

New instruction	Action data tables used in recombination (given in the form of instructions)	Successful cut & recombine	Successful baseline	Remarks
Rotate cup on table.	(1) Rotate rotor axle.	10/10	–	
Take jar and *place* in box.	(1) Place jar in pot.	10/10	9/10	In compiled version, box was slightly pushed twice.
Take spoon from bowl and *insert* into jar.	(1) Take spoon from bowl and drop into box. (2) Insert knife into jar.	9/10	–	
Take cup from table and *put over* fixture.	(1) Put rotor cap over rotor axle.	9/10	–	
Lay jar on tray.	(1) Put jar into pot. (2) Take bottle from tray and lay on table.	6/7	–	Snippet ranked third was chosen by expert for "lay" movement.
Shake measuring beaker and put it on tray.	(1) Take measuring beaker from table and put on tray. (2) Take jar from tray, shake, and put on table.	4/5	0/5	Large improvement with respect to baseline.
Unscrew lid from thermal mug.	(1) Unscrew lid from jar.	2/3	2/3	One of two equally ranked snippets had to be chosen.
Drop bottle into wastebasket.	(1) Take bottle from tray and drop into box. (2) Drop rotor cap into box. (3) Drop bottle cap into wastebasket.	6/10	0/10	Large improvement with respect to baseline.
Push bottle *away* from jar.	(1) Push bottle away from box. (2) Push cup away from jar.	6/10	6/10	For reliable execution this action needs two object denominators.
Invert a jar.	(1) Pick jar and place into pot. (2) Invert bottle cap.	3/6	-	

is also a feed-forward method, which, however, does not consider object properties. By contrast, in the cut & recombine approach, we reuse ADT snippets based on both action *and* object similarity. Unlike this, in the baseline method (Aein et al., 2013), each individual action is defined using one set of parameters tuned by trial-and-error for kitchen-sized objects (cups, bowls, bread, fruits, etc.). For example, the grasp primitive in this library uses a wide pre-grasp in order to increase the success of grasping most of the mentioned objects in uncluttered scenes. To give another example, to lift the main object in the *place* action, a specific fixed lifting height of 15 cm is used. Thus, the comparison

Table 10. Existing action data tables.

No.	Action	Main object	Primary object	Secondary object
1	Drop	Bottle cap	Tray	Wastebasket
2	Drop	Bottle	Shelf	Box
3	Drop	Bottle	Tray	Box
4	Drop	Pressure ring	Support	Cup
5	Drop	Rotor cap	Table	Box
6	Drop	Spoon	Plate	Box
7	Insert	Knife	Table	Jar
8	Invert	Bottle cap	Table	Table
9	Lay	Rotor axle	Shelf	Tray
10	Place	Bottle	Table	Bottle holder
11	Place	Bottle	Table	Cup
12	Place	Bottle	Table	Pot
13	Place	Jar	Table	Pot
14	Place	Jar	Table	Bottle holder
15	Place	Jar	Tray	Shelf
16	Place	Rotor cap	Conveyor	Fixture
17	Place	Rotor cap	Conveyor	Robot platform
18	Place	Rotor cap	Fixture	Table
19	Place	Measuring beaker	Table	Tray
20	Push apart	Bottle	Box	Table
21	Push apart	Cup	Jar	Table
22	Put over	Rotor cap	Table	Rotor axle
23	Shake	Bottle	Tray	Tray
24	Shake	Jar	Tray	Table
25	Rotate	Rotor	Table	Table
26	Unscrew	Bottle cap	Bottle	Table
27	Unscrew	Bottle cap	Bottle	Tray
28	Unscrew	Jar lid	Jar	Table

of cut & recombine with the object-independent approach allows us to determine whether taking objects into account increases the success of robotic execution.

As discussed, the basis for our experiments was a database of 28 ADTs for 10 different actions performed with different objects (twelve different main, nine primary, and eleven secondary objects), see Table 10.

The instructions for the test set were chosen so that similar actions and similar objects could be found in the ADT database, but in different combinations. Also, we included some examples where the object mentioned in the new instruction was never dealt with before, to investigate whether generalization could work in those situations, too.

To obtain statistics on execution success, we performed the same instruction with different object combinations and different object placements in the scene (see object sets used in different actions in Figure 7). Our target was ten different object or placement settings for each instruction, however, if only a single object type was mentioned in the instruction, we performed the instruction only as many times as we had different objects (e.g., we had only five different measuring beakers and only three different thermos mugs). With these object choices, we tried to push our algorithm to the limits, using considerably larger and smaller objects than in the execution examples in the ADTs, but we did not use objects or object configurations where it was clear in advance that the algorithm would not be able to handle the situation at all (e.g., objects differing in size by orders of magnitude as compared with the examples or objects touching each other when the example objects were standing separately).

The results for the cut & recombine approach are shown in columns two and three in Table 9. Column two shows from which ADTs information was recombined (for ease of reading, ADTs are given in form of instructions). In column three, we show how many trials were successful; the overall numbers of performed trials are given behind the slash.

As column two shows, in four out of ten cases, the subsymbolic processing has chosen to perform new actions using snippets from a single ADT (without recombination), but in the remaining cases snippets from several different ADTs were recombined. On five occasions, the new ADT was recombined using snippets from two ADTs; one case occurred where the new ADT was recombined from three different ADTs.

Instructions in the table are sorted from most successful to least successful (column 3). For two instructions, the success rate was 100%, for another five instructions, errors

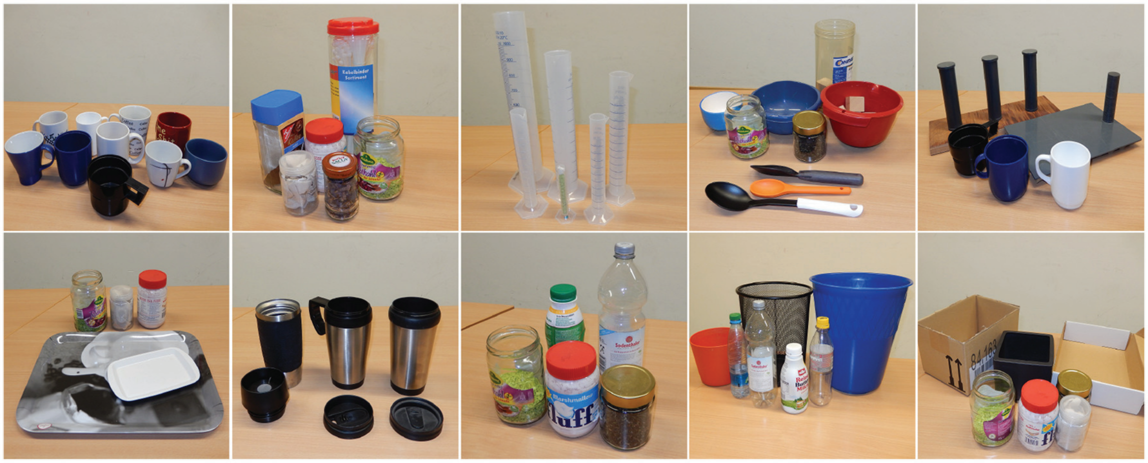


Fig. 7. Object sets used in the experiments.

happened only once, and there were three instructions where errors happened in a systematic way (the three last rows in Table 9).

Example videos for successful and unsuccessful executions can be found on the website.[3]

Next, all results will be discussed in some detail to allow the reader to judge performance and to show that simple error correction methods would almost always suffice to resolve the remaining errors of the feed-forward cut & recombine approach.

The two instructions that were 100% successful were "Rotate cup on table", and "Take jar and place it in box". The reuse and execution success of those instructions are analyzed in more detail next:

- For rotation of the cup, movements were taken from a single ADT describing rotation of a rotor axle. Although the object in the existing ADT was quite different, the action, as such, was very simple and the replacement worked.
- For placing the jar in a box, the movements were again taken from a single ADT. The execution was successful in the sense that the jar ended up in the box in all experiments, but the box was slightly pushed two times. This happened because we were defining movement only through the relative object center coordinates and the size of objects was not considered. However, lifting over the rim of the box also depends on the size of the box. Where the box was taller or wider than the object that was used when making the ADT from which the snippets were being reused, the approach toward the box became tighter. This was why the side of the box was touched by the jar and the box was slightly moved.

The five instructions where incorrect execution only happened once were "Take spoon from bowl and insert into jar", "Take cup from table and put it over fixture", "Lay jar on table", "Shake measuring beaker and put it on tray", and "Unscrew lid from thermal mug". Individually, those executions are analyzed next:

- Insertion of a spoon in a jar was recombined using snippets from two ADTs. Approach, grasping, and lifting of the spoon were taken from the ADT describing dropping of the spoon, while the insertion movement was taken from a different ADT, describing insertion of a knife in a jar. The execution worked successfully, except in one case of a very tight fit between the jar and the spoon, where the spoon got stuck at the mouth of the jar, owing to slight pose inadequacy.
- Putting the cup over the pin of a fixture depended on a single ADT describing putting a rotor cap over a rotor axle. These objects were similar in size to the cups and fixture pins in our new scenario. The only unsuccessful execution occurred with a very small cup. The movement for "putting over" in the ADT had the approach slightly sideways, with a small offset between the center of the fixture and the center of the rotor cap that was put over, as would be the case if the fit is tight. Such a movement did not work if the cup was short.
- Laying of the jar was recombined from two different ADTs, where approach and grasp were taken from the ADT of placing a jar and the specific laying movement was taken from another ADT for laying a bottle. In laying the jar, only one unsuccessful execution happened; this was because of uneven mass distribution in a jar (heavy sand on the bottom); the jar flipped back into a standing position from almost a lying position when the robot hand released it.
- Shaking the measuring beaker was also recombined from two different ADTs; approach and grasp (from a side) of the measuring beaker were taken from one ADT, while the shaking movement was taken from the other ADT, where the grasp contained in the same ADT would have been incorrect, as the jar therein was grasped from above, which does not work for the

measuring beaker. The execution of the instruction was correct, except for the smallest measuring beaker, where shaking was successful but stable placing on the table was not guaranteed.
- For unscrewing the lid from the thermos mug, we encountered the aforementioned situation that there were two equally ranked ADT snippets. The user had to choose: either snippets could be taken from unscrewing a bottle or from unscrewing a jar. Here, the jar case was chosen for reuse, as from human knowledge the radius of the jar lid is more similar to the radius of the thermos mug lid. One execution was unsuccessful because one example of the thermos mug had a lid that fit very deeply and tightly into the mug; this did not match the conditions of unscrewing the jar.

Several systematic errors occurred in three cases: “Drop bottle into trash basket”,“Push bottle away from jar”, and “Invert a jar”. The reasons for this are given in detail next:

- Dropping of the bottle into the wastebasket was recombined from three different ADTs: dropping the bottle into a box (for reaching and grasping), dropping the rotor cap into a box (for lifting off the table) and dropping the bottle cap into a wastebasket (for the specific drop motion). We had four incorrect executions, for the following reasons: grasps were not stable on three occasions; twice when grasping a substantially bigger bottle and once for a substantially smaller bottle, as compared with the bottle for which the ADT was recorded. The bigger bottles could not be grasped stably because we were taking the pre-grasp width from ADTs and thus the pre-grasp was adjusted to a smaller bottle. A tight pre-grasp in an uncluttered scene proved disadvantageous, alternatively it would be able to serve its role in a cluttered scene. The fourth unsuccessful case happened for a substantially smaller wastebasket that was used as compared with the one with which the ADT had been made and the bottle fell just behind the rim. In summary, unsuccessful executions only happened here when we were “provoking” our framework with objects of substantially different geometries.
- Pushing the bottle away from the jar was recombined from two different ADTs: pushing the bottle away from the box and pushing the cup away from the jar. Thus, the general targeting of how to push the bottle was correct; however, the entire action was performed correctly for only a subset of object configurations. This time, the limitations of our model were at fault: we only used one object denominator per movement primitive, while two object denominators are needed to define pushing away, as the hand needs to go between the objects.
- Inverting a jar was combined from two ADTs: one for approaching and picking up the jar and the other one for inverting a very different object: a bottle cap. In spite of such large object differences in the ADT used, in all six cases, picking up and inversion went error-free, where the three errors only happened as the inverted jar did not land stably on the table. This resulted from our robotic implementation, where placing of an object was not elaborate (not force or otherwise controlled).

For five out of ten instructions, we could make comparisons with execution based on the object-independent action library (baseline method). Note that we only made comparisons in cases for which the corresponding action was already available in the object-independent action library prior to this study. We did not expand the library by developing new actions specifically for comparison with our current work, as the library uses heuristic approaches and new actions would thus have come out biased toward the examples used in the current work, possibly leading to excessively favorable comparisons.

In the comparison with the baseline we twice saw substantial improvement by using our cut & recombine approach (for the instructions “Shake measuring beaker and put it on tray”, and “Drop bottle into wastebasket”), and similar performance three times. Note that an improvement was not achieved in all cases, because the object-independent action library already performed well on some of our objects. This is because the library was tuned for performing table-top manipulations with a wide set of objects, including a subset of the objects that we used in our experiments. However, in those cases where object-specific handling was needed, large improvements were achieved when using ADT recombination. In the case of shaking the measuring beaker, the improvement came from object- specific reaching and grasping from one side, as indicated in the first of the recombined ADTs, where the grasp from the top inscribed in object-independent actions for the measuring beaker was completely unusable. For dropping an object into a wastebasket, the existing ADT gave an example similar to how a human being would perform the action: dropping from relatively high above. This allowed for variability in trash basket and bottle sizes. By contrast, the action in the object-independent library was defined as dropping with approaching the small container tightly, which was not similar to a trash-dropping situation and never worked.

It is also interesting to note that there was only one case (pushing the bottle away from the jar), which pointed to a possible problem of the cut & recombine approach for this given case. (We had only used one object denominator per movement primitive but two object denominators are needed to define pushing away.) All other cases of error can be corrected by feedback error correction mechanisms. Thus, it seems that, for most of the existing table-top operation, the framework is already relatively complete and useful.

We had only a six out of ten (60%) success rate in the execution of the command “Drop bottle into wastebasket”, based on the symbolic-name-based snippet ranking procedure. As the errors mainly happened because of incorrect

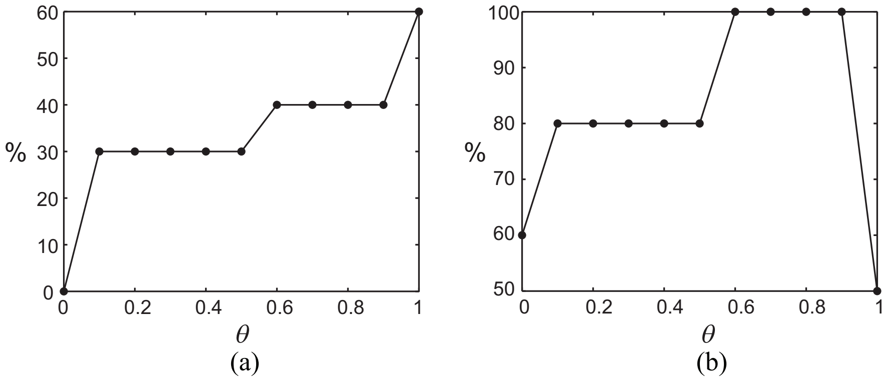


Fig. 8. Performance evaluation using a hybrid ranking procedure. (a) Percentage of movement primitives replaced using the hybrid procedure as compared with movement primitive composition obtained using symbolic-name-based ranking procedure. Results are given for different weights θ (equation (1)), indicating the influence of the sub-symbolic counterpart in ranking. (b) Execution success as a function of θ.

object sizes in the selected ADTs, for this command, we repeated the experiments using the hybrid ranking procedure described in equation (1). In this case, we had different sets of movement primitives (extracted from different sets of ADTs) for the same command but different scene instantiations, depending on the size and aspect ratio of the presented objects. As some errors in the original execution of the command came from the robot not being able to grasp a large bottle, we had to introduce a new ADT, as there were no examples in the original set of ADTs presented in Table 10 on manipulating large enough objects with the denominator "main". We added an ADT for the command "Take a bottle from a tray and drop it into the box" (ADT No. 29), but this time with a larger bottle than that in ADT No. 3 (Table 10).

To evaluate the performance of the hybrid ranking procedure, we changed the weight θ in equation (1) in the interval $[0, 1]$ and investigated how the composition of the movement primitives and the success rate of execution changed. The proportion of changed movement primitives over the entire pool of 10 scene instantiations ($\times$ 4 movement primitives per instantiation) with θ in the interval $[0, 1]$ is provided in Figure 8(a). One can see that the percentage of changed movement primitives increases with θ and reaches 60% when $\theta = 1$ (only object size matters). The execution success rate as a function of θ is provided in Figure 8(b). For pure symbolic ranking, we have a 60% success rate. When combining symbolic and sub-symbolic information in the ranking procedure, the success rate increases and reaches 100% for a wide range of parameters θ. When ranking is based on object size alone, the success rate is 50%.

Example movies for $\theta = 0.7$ showing successful executions in two cases: a large bottle and a small (short) bottle, where grasps were unsuccessful in our previous approach, can be found on our website.[4] Now the large bottle is grasped successfully, because of a wider pre-grasp and the small bottle is grasped differently, namely, from above. This grasp comes from a different ADT, now ranked highest using the hybrid similarity measure.

Clearly, the performance of the cut & recombine approach largely depends on the existing ADT database. If the ADT database were different, the results would most probably be different too. The interplay between the new instruction and the ADT database is quite complex, already, given the 28 ADTs (29 in the second part of the study), we could not predict what our algorithm would choose to recombine. Thus, the situations we have provided were not staged.

Summarizing, the results show the general possibility of ADT snippet recombination and ADT-based action transfer. There are cases where usage of object-dependent ADTs brings substantial improvement in comparison with object-independent actions, as defined in our baseline method. The observed deficiencies in our transfer procedures, such as an insufficient number of object denominators in the case of the push-away action, indicate a path for improving our framework.

7. Discussion

We have proposed a framework for existing action component reuse in new robotic execution examples. The task is conveyed to the robot by language instructions. Language instructions are parsed for symbolic names of actions and objects. These symbolic names then allow a set of previously executed instructions containing potentially reusable code components to be found. We cut and recombine code snippets within this set to obtain code for execution of the new instruction. By "code", we mean parametrized transcripts of previous executions presented in XML structures called action data tables (ADTs).

Our framework allows robots to be programmed by instruction to perform table-top operations, which do not require great precision (e.g. in a kitchen scenario), where examples of the execution of *similar* instructions exist.

However, owing to the code recombination that we perform, we do not require that exactly the same instruction was executed in the past.

The specificity of the proposed approach lies in the strict temporal structure of the analysis of temporal neighborhoods at two hierarchical levels of temporal chunking. Neighborhoods are defined using (1) SECs and (2) movement primitives in the action templates. As discussed in Subsection 3.4, we constrain the language in the instructions so that omissions in instructions at the defined level of granularity are not allowed. In this way, we achieve a functioning instruction-to-ADT link. It is mainly through all these action structuring efforts that code snippet recombination becomes viable.

Through the recombination processes, new ADTs are created. Thus, this approach ultimately creates a continuously growing database.

7.1. Parametric considerations

There are several entities on which the results of our procedure depend. One important entity is the neighborhood of movement primitives. We parametrize the neighborhood by its “width”, which in our study is defined by a pair of parameters: the number of neighboring SEC states and the number of neighboring movement primitives included in the neighborhood. The “width” influences how many other movement primitives from other actions are allowed to replace movement primitives of the new action. When one considers a “wider” neighborhood, fewer hits are found in different actions, and vice versa. In our study, we set the neighborhood (see Algorithm 2 in the appendix) by expert review of the suggested replacement lists and choosing the most suitable widths. End-performance-based evaluation of different neighborhoods is also possible, but in this study it was not performed because this requires a lot of effort.

Another set of parameters is associated with the replacement ranking rules, based on action and object names. We have heuristically chosen the ranking order, based on the assumption that the object on which the movement primitive depends (i.e., the one that stands in the object denominator) is more important than the other objects.

Finally, ADT interpretation plays an important role. Currently, we are only considering object positions defined by the center of the object bounding box as well as pairwise object relations (TCP to main object, main to primary object, and main to secondary object). More detailed interpretations, where object size is taken into account or relations between more than two objects in the scene are considered, would form alternative methods, which were not yet considered.

7.2. Comparison with the state of the art

Our approach is related to a group of studies attempting to bridge the gap between natural language and robotic action (Bollini et al., 2013; Lisca et al., 2015; Misra et al., 2016; Tellex et al., 2011). We, in fact, address a question that is slightly narrower: programming robots by instruction, which puts limitations on the language provided to the robots. We use language only as a means to define an instruction and analyze only instructions indicating “robotic actions” (the actions for which we have made a formal action description). We forbid essential omissions in instructing: all actions within the defined granularity must be spelled out explicitly. These constraints allow us to achieve a relatively lightweight approach, as compared with the previously mentioned systems attempting full natural language complexity. Next, we will compare our work with those approaches in more detail.

Misra et al. (2016) investigate the translation of natural language instructions into a sequence of predefined robot-executable routines. They use an energy function that encompasses natural language evidence as well as environmental evidence and attempt to find the maximum likelihood solution based on learning examples. The approach (unlike ours) requires many learning examples and is firmly based on a set of predefined robot-executable routines. Changing the sample of those routines or changing the domain would require extensive relearning, while our approach works by adding new action templates and new execution examples to the existing database as needed and can thus be re-adapted to a new domain in a continuous manner. In addition, our approach processes language and sub-symbolic entities in two different processing steps, thus making the method easier for a human operator to understand and access. Moreover, our approach suggests strict structuring of robotic actions, defining granularity and a two-level hierarchical composition, while the approach of Misra et al. (2016) uses an inconsistent sample of robot-executable routines at different levels of granularity (e.g., compare MoveTo (simple) and Open-Close doors (complicated), an example taken from Table 2 in Misra et al. (2016)). Our more structured view of robot executables could be advantageous in this (and similar) data-driven approaches. It is, however, clear that our approach cannot handle some of the aspects that are central to the approach of Misra et al. (2016). We do not handle missing instructions and we do not reason about object states, both of which requires general-purpose commonsense knowledge, which is not in the center of our study.

A group of approaches exist for handling strongly task-specific natural language instructions. One example is presented by Lisca et al. (2015), who analyze how to transform natural language instructions for chemical experiments into robot control programs. These authors address much more complicated instructions than those in our study (an example being “Neutralize 75 ml of hydrochloric acid”.). This, however, requires extensive hand design, which is supplemented by training of a Markov logic network based on datasets collected specifically for the developed core structure. By comparison, in our approach, we only design the action template for each action, as shown in Table 2 for action *place*. (Note that the SEC in the table is not hand-designed but denotes an objective sequence of touches and un-touches of objects throughout manipulation.) Plans for actual robotic action in Lisca et al. (2015) (e.g., aspirating a pipette and pipetting in the "neutralization" action) are manually linked during the design of the core structure. Automated inference processes, which are emphasized in this study, are possible only after hand design and learning steps have been accomplished. A similar approach was applied for executing cooking recipes, specifically, pancake making, by Nyga and Beetz (2012). Reuse of previous action components, which lies at the center of our study, is not addressed in the cited studies. There, the data structure for a different instruction must be hand-designed essentially from scratch.

Another example closely related to the domain is the cooking robot developed by Bollini et al. (2013), who specifically address the execution of baking recipes. Translation of natural language instructions for baking is done based on hand-annotated mapping of natural language recipes into so-called cooking primitives and then mapping those primitives into robotic actions. For example, the cooking primitive "bake" is mapped into a long sequence of robot-executable primitives of operating the oven, such as opening and closing doors, inserting a dish into the oven, and so on. In this study again, instructions are given at a higher level than in our study but with the need for substantial hand design and learning based on human-annotated examples. Reuse of structures in this framework is not foreseen. In our approach, the robot is instructed at a lower level, that is, immediately indicating robotic actions. In this way, we overcome the requirements for complicated hand-designed structures as well as the difficulties of changing domains, as the number of actions, at the granularity we are at, is limited (see Wörgötter et al. (2013) for discussion). We still, however, manage to introduce reuse of previously executed action details.

One domain, which is easier to describe using natural language, is navigation. There one can find more straightforward grounding of natural language instructions in real-world geometry; thus this domain can be tackled quite successfully (Artzi and Zettlemoyer, 2013; Guadarrama et al., 2013; Kollar et al., 2014; Matuszek et al., 2013; Rosenthal et al., 2016; Tellex et al., 2011; Walter et al., 2014; Williams et al., 2017). Navigation can be supplemented by, for example, "pick and place" or "point" actions, where geometrical considerations define actions up to trajectory precision (Guadarrama et al., 2013; Tellex et al., 2011). Kollar et al. (2014) analyze a wider set of verbs, referring to more complicated actions like "follow", "meet", or "bring'; however, those verbs are still groundable through path description. We are working in the manipulation domain, going beyond strongly geometry-bound actions, which require richer grounding approaches. We have chosen example-based reuse of action components instead of building more rigorous world (or path) models, which is the approach used in the cited studies.

When instructing a robot using natural language, dialog systems can help resolve ambiguities in the instructions. Work in this direction exists (Gemignani et al., 2015; Perzylo et al., 2015; She et al., 2014; Thomason et al., 2015).Though we do not address the issues of disambiguating instructions through dialog, we have met, in our work, many of the difficulties indicated by Perzylo et al. (2015), such as difficulties in disambiguating verb senses. We also resolve these questions by querying the user, e.g. when the algorithm cannot determine the central action in our framework.

A natural language interface is a convenience but not absolutely necessary to employ our suggested sub-symbolic recombination procedure. For example, robot programming on predefined action blocks (Alexandrova et al., 2015; Schlette et al., 2014) or skills (Bøgh et al., 2012; Steinmetz and Weitschat, 2016; Stenmark et al., 2015) could also be used as an interface. All the cited approaches are also designed to help non-experts to program robots. Stampfer and Schlegel (2014) and Wächter et al. (2016) go further and define reusable state charts for easier humanoid robot programming, where the state chart defines an action as a branching structure of states. However, in all these approaches, the user is expected to parametrize predefined blocks (or state charts) from scratch and reuse is addressed only in a limited way. For example, Alexandrova et al. (2015) suggest generalization to different numbers of objects by embedding the predefined blocks in loops or adjusting a block developed for grasping to different size of objects by readjusting thresholds. Wächter et al. (2016) show a use-case of transferring a state chart to a different robot. Stenmark et al. (2015) talk about skill reuse, but only in the sense of reparametrization of the previously developed skill by a user, without providing an explicit framework for choosing the skills for reuse. By contrast, our study addresses the reuse of previous examples of execution based not only on action but also on object similarity, with the aim to reduce the user's parametrization effort.

At the other end of the spectrum of robot programming stands objectcentric programming, where the scene appearance is used to derive robot code. In objectcentric programming, geometric and relational properties play a central role in defining the action (Angerer et al., 2009; Hart et al., 2015; Huang et al., 2015; Perzylo et al., 2016). We adhere to this approach at the low (movement primitive) level, reusing relations between, for example, main and primary or main and secondary objects, instead of planning an action from scratch, as suggested in the cited studies.

An intermediate approach with constrained natural language use in robot programming was analyzed by Stenmark and Nugues (2013). In this approach, only verbs corresponding to skills are allowed; thus language is not fully natural, similar to our approach (we, however, also handle synonymy). The cited approach allows fewer actions and does not address our question of how to best parametrize skills based on previous execution examples.

Instead, a skill sequence with default parameter values is given to a programmer for parameter adjustment.

Alternative studies do exist that emphasize data collection from robotic experiments in both industry-oriented (Björkelund et al., 2011; Persson et al., 2010) and service robotics domains (Beetz et al., 2016; Ovchinnikova et al., 2015; Riazuelo et al., 2015; Tenorth and Beetz, 2013; Tenorth et al., 2013; Winkler et al., 2014). In the following, we will discuss how the mentioned approaches relate to our study. Persson et al. (2010) and Björkelund et al. (2011) mainly address the question of how to convert code adhering to the emerging industrial standard AutomationML (Drath et al., 2008) into RDF representations allowing reasoning (Miller, 1998) and accumulate data adhering to Semantic Web standards (Shadbolt et al., 2006). These initial efforts are followed by limited reuse attempts (Stenmark and Nugues, 2013; Stenmark et al., 2015), but not in a Semantic Web context. Persson et al. (2010) and Björkelund et al. (2011) had already found that the desired conversion into RDF structures has obstacles; in industry the code is currently mainly accumulated in AutomationML formats. AutomationML, however, does not target the specificity of defining reusable robotic actions or action fragments, as in our study, but rather targets standardized coding conventions, such that code is easily reusable by another programmer.

Another group of studies addresses the question of accumulating robotic knowledge in service applications (Beetz et al., 2016; Bozcuoğlu et al., 2018; Riazuelo et al., 2015; Tenorth and Beetz, 2013; Tenorth et al., 2013; Winkler et al., 2014). This line was started with the RoboEarth project to create a "World-Wide Web for Robots" (Waibel et al., 2011). There the reuse of knowledge was investigated from a very wide perspective, where the aim was to accumulate "all" information required for the robot: action recipes (tasks), actions, object models, environment maps, algorithms that were used for creating accumulated data, robot capabilities required to perform actions, and so on (Tenorth and Beetz, 2013; Tenorth et al., 2013; Winkler et al., 2014). Owing to its complexity, this approach can no longer be transferred to the users as a collection of algorithmic ideas, but only as a program package; indeed, it is released as an open-source ROS package. However, again because of complexity and the large amount of special knowledge required for each example, only a few application examples with full functionality of the RoboEarth system have been demonstrated so far. We propose a less powerful but more accessible approach of storing and reusing robot experience. For example, we do not handle the issues of robot capabilities and just talk about table-top manipulations with a "standard" arm-hand system. We require the robot to be instructed at a much lower level, as compared with the task level (e.g., "Serve a drink") given by Tenorth et al. (2013). However, we do not require complex high-level knowledge from the robot about tasks existing in the (human) world. Also, our approach appears more compact and more rigorously hierarchically structured. Thus, the two approaches are very different and there can be no single answer as to which approach is more applicable, as this depends on the task and circumstances.

Follow-ups of these studies do not suggest using the entire system, but rather parts of it, specifically by means of cloud services for the robotic community (Beetz et al., 2016; Riazuelo et al., 2015). However, none of these studies looks at the same aspects as we do. Riazuelo et al. (2015) concentrate on robots reusing knowledge about similar environments, for example, hospital rooms, and how to find objects in those rooms. Here geometric considerations are primarily used to adapt existing knowledge to the new situation and robotic manipulations are only considered in connection with discovered object positions. A similar question of memorizing common object locations is addressed by Ovchinnikova et al. (2015). Beetz et al. (2016) do indeed talk about manipulation data reuse, but for a different purpose than in our study. They offer to use the collected data for the analysis (in a sense of reconsideration) of previously performed experiments, for example, through creating datasets for analyzing errors in perception algorithms. Similar aims (error analysis and statistics) are indicated in a previous study (Niemueller et al., 2012). In none of these studies do the suggestions for data reuse go into the direction of "programming" new instructions based on previous programs of similar instructions. Conversely, we are specifically suggesting a framework for data reuse for new instruction coding and execution.

Possibly the most closely related work is that of Bozcuoğlu et al. (2018), who use episodic memories in addition to domain knowledge. A number of robot execution examples are collected and grasps and, to some degree, trajectories are transferred to a different setting. This study, however, concentrates on a single (though difficult) action: opening hinged containers (specifically, refrigerators). Knowledge transfer rules are fully adapted to hinge-joint doors with elongated handles but do not generalize to other actions or objects.

Another quickly growing field where robot experiences are implicitly reused concerns deep learning on visuomotor data (Finn and Levine, 2017; Ku et al., 2017a; Levine et al., 2018). While Finn and Levine (2017) and Levine et al. (2018) learn to predict the consequences of pushing and grasping motions in an end-to-end manner, Ku et al. (2017a) use a more structured approach by associating selected convolutional neural network features obtained from scene analysis to robot actions. All mentioned approaches, though not requiring much human supervision, are tuned to solving specific situations: performing pushing to a predefined location (Finn and Levine, 2017), grasping of objects residing in front of a robot (Levine et al., 2018), and grasping of a power drill (including pushing or dragging when required) (Ku et al., 2017a). Thus, though promising, these approaches show for less action variety than the approach used in our study. The end-to-end learning approaches (Finn and Levine, 2017; Levine et al., 2018), though able to handle a large variety of objects, are, however, much more expensive in terms of robot experience than our approach. Moreover, they are not easy to extend (e.g., if one needs not grasping as such, but grasping for insertion of an object into a narrow container, this would require full retraining). By contrast, our system only needs a few relevant examples for extension to a new task.

Finally, one could claim that examples of previous robot experience are not needed, as better methods are being developed for motion (Latombe, 2012; Sucan et al., 2012) and grasp (Alterovitz et al., 2016; Bohg et al., 2014; Lippiello et al., 2013; Vahrenkamp et al., 2012; Vezzani et al., 2017) planning. However, reusing previous experience of similar tasks can constrain planning in a useful way; thus, such approaches can be used together with planning.

To compare our study with different approaches discussed above in a more rigorous manner, we have compiled Table 11, which indicates the properties of 20 representative studies.

All studies in Table 11 describe large robotic systems supported by different knowledge acquisition components, including learning using neural networks, using external (Internet) resources, and programming reusable components. Some of these systems are specifically designed only for a certain task, like grasping (in a bin-picking setting), opening hinged doors, or performing well-defined cooking sequences (Bollini et al., 2013; Bozcuoğlu et al., 2018; Levine et al., 2018), leading to a high degree of domain restriction. By contrast, our approach is, in principle, not domain restricted and ADTs could also be developed for different tasks from the here-demonstrated manipulation domain. Our study also differs from the group of domain-unlimited approaches, which are mostly made for manual reuse of previously defined control structures (Alexandrova et al., 2015; Schlette et al., 2014; Stenmark et al., 2015; Wächter et al., 2016): we specifically consider how to find and combine reusable components in an *automated* way.

Only a few systems have been rigorously (statistically) evaluated on a robot and many studies perform simulations, sometimes paired with proof-of-concept (PoC) robotic experiments. Furthermore, most studies use very few objects and actions (fewer than 10 each), whereas our evaluation used 10 actions and 45 objects. Generally, studies that use manual (human-programmed or human-guided) approaches for knowledge acquisition can cope with small amounts of data for setting up their system, whereas (semi-) automatic approaches (e.g., deep learning) usually rely on large databases. Our approach starts with few data and offers lifelong automatic extension, not found in any of the other automatic systems. Several methods are very strong in handling symbolic information (for example handling missing information in an instruction), which we do not attempt with our system to keep the data requirement low. Conversely, none of the approaches is able to automatically recombine sub-symbolic information from different objects or actions, which is a unique feature of the here-presented method. This analysis shows that many different approaches currently exist, which, owing to their specificities, are not easy to compare. In addition, quantitative comparative evaluation is impossible, due to their complexity and their domain restrictions.

7.3. Future work

While this study has proven that the principle of cutting and recombining ADT snippets to obtain execution information for new instructions works as such, more work is needed to define ways of using information in the case of a much larger ADT database. When the same or a similar action is performed with similar objects in several contexts, much more context information can be used in addition to that used in this study. In this way, we could find more appropriate snippets and approach a stage where the robot can operate with very little human interference.

One could raise the question of whether the method would scale. We argue that it is possible to develop a good indexing system, which would allow appropriate examples to be chosen quickly from large execution transcript databases. The development of such indexing systems very much depends on further developments of the theory of robotic actions, to which we have contributed in our earlier works (Aksoy et al., 2011;Wörgötter et al., 2013) as well as here (specifically, by defining action templates as important action structuring elements, as well as in initial attempts to define ranking rules for action snippet similarity). However, more elaborate systems for action similarity evaluation and action knowledge transfer could be introduced for an extended ADT treatment. Using deep neural network features for sub-symbolic information representation as, for example, introduced by Ku et al. (2017a,b) may be promising.

Currently, our work only addresses parametrization and reuse in feed-forward action representations. Introducing feedback and error correction schemes is another branch of continuation for this work.

In addition, a more advanced structure and use of the *instruction ontology* should be considered. First, similar instructions can be ranked, including not only synonymy but also distances in the ontology tree, especially for objects. Object properties, as well as object part considerations can also be handled through the ontology; this then needs to be treated in ADT-based processing, as well. Concerning the action counterpart of the ontology, one could keep instruction sequence information, which would allow the system to treat cases of missing instructions statistically based on information from previous instruction sequences accumulating in the ontology over time. This would release the constraints on instructing the robot.

Finally, to evaluate the work of such a system in practice, research on human interaction with the system would be required. Human ability to formulate instructions in a sufficiently precise manner, human abilities to choose appropriate snippets from the ranked lists, and different human

Table 11. Comparison of different studies.

	Task	Evaluation procedure	Instruction format	Knowledge acquisition	Amount of data used to set up the system	Domain restriction	Number of different actions in evaluation	Number of different objects in evaluation	Handling of missing instructions/ object info	Reuse of motioncontrol parameters	Reuse of motion from different actions	Object-specific reuse of motion components	Rule system for action component reuse	Human-comprehensible format	Extensions to the system
Björkelund et al, 2011	Manip.	None (1)	Coding	Programg.	Not defined	N	3 (41)	Not specified	N/N	Manual	N	Manual	N	Y (74)	Manual
Tellex et al, 2011	Navig. & pick & place	Simulat. (2)	Nat. language	Learning (10)	Medium (16)	Y (30)	2 (42)	3 classes (59)	N/N	N	N	N	N	N	Retraining
Bøgh et al, 2012	Mobile manip.	Robot prelim.	Demo.	Programg.	Medium (17)	N	2 (43)	2 (60)	N/N	N	N	N	N	Y	Manual
Bollini et al, 2013	Mobile manip.	Mostly simulat. (3)	Nat. language	Learning (11)	Medium (18)	Y (31)	3/9 (44)	1 class (61)	N/N	N	N	N	N	N	Retraining
Guadarrama et al, 2013	Visual perception & manip.	Robot (4)	Nat. language	Learning (12)	Medium (19)	Y (32)	5 (45)	50 classes	N/N	N	N	N	N	N	Retraining
Tenorth et al, 2013; Tenorth & Beetz, 2013	Mobile manip.	Robot (PoC) (5)	Task command	Many exter. sources (13)	Large (20)	Y (33)	1 task (46)	3 (62)	N/Y	N	N	N	Y (69)	Y (75)	Redefinition of several components
Stenmark & Nugues, 2013	Manip.	Robot (PoC)	Finite language	Programg.	Not defined	N	4 (47)	3 (63)	Y (68)/N	Grasp reuse	N	Grasp reuse	N (70)	Y	Manual
Kollar et al, 2014	Navig.	Simulat. & robot	Nat. language	Learning (14)	Small (21)	Y (34)	5 (48)	Not specified	N/N	N	N	N	N	N	Retraining
Schlette et al, 2014	Manip.	Simulat. (PoC)	Visual blocks	Programg., macronizing	Not applicable	N	3 (49)	Object indep.	N/N	Manual	N	N	N	Y	Manual
Lisca et al, 2015	Manip.	Robot (PoC)	Nat. language	Learning & data bases (15)	Not indicated	Y (35)	2 (50)	One set (64)	N/Y	N	N	N	N (71)	Y (75)	Redefinition & retraining
Alexandrova et al, 2015	Navig., manip. & act. percep.	Robot (PoC)	Visual blocks	Programg., macronizing	Small (22)	N	6 (51)	4 (65)	N/N	Manual	N	Manual	N	Y	Manual
Stenmarket al, 2015	Manip.	Robot (PoC)	GUI or coding	Programg.	Not defined	N	1 task (52)	2 (66)	N/N	Manual	N	CAD models	N	Y	Manual
Perzylo et al, 2016	Manip.	Robot (PoC)	Object-centered GUI	Programg.	Not indicated	N	2 tasks (53)	Not specified	N/N	Manual	N	CAD models	N	Y	Manual
Wächter et al, 2016	Mobile manip.	Robot (PoC)	GUI or coding	Programg.	Small (23)	N	3 tasks (54)	Not specified	N/N	Manual	N	N	N	Y	Manual
Misra et al, 2016	Mobile manip.	mostly simulat. (6)	Nat. language	Learning	Medium (24)	Y (36)	5/13 (55)	Multiple (no. not given)	Y/Y	N	N	N	Y (72)	N	Retraining
Ku et al, 2017	Manip.	Robot	None (8)	Deep learning	Small (25)	Y (37)	3 (56)	Power drill	N/N	Implicit	N	One object only	N	N	Retraining of CNN
Finn & Levine, 2017	Pushing	Robot	Location pointer only	Deep learning	Large (26)	Y (38)	1 (38)	Many	N/N	Implicit	N	N	N	N	Retraining of CNN
Levine et al, 2018	Grasping	Robot	None (9)	Deep learning	Large (27)	Y (39)	1 (57)	Many	N/N	Implicit	N	N	N	N	Retraining of CNN
Bozcuoglu et al, 2018	Mobile manip.	Robot (7)	Not defined	Partial learning & ext. sources	Small (28)	Y (40)	2 (40)	1 (67)	N/N	Partial	N	Y	Y (73)	Y (75)	Redefinition & retraining
Our	**Manip.**	**Robot**	**Simple nat. language**	**Learning from prev. executions**	**Small (29)**	**N**	**10 (58)**	**45**	**N/N**	**Y**	**Y**	**Y**	**Y**	**Y**	**Acquisition of new ADTs**

ADT: action data table; CNN: convolutional neural network; PoC: proof of concept.

[1]Architectural framework (no experiments). [2]Evaluation with the help of user study (Amazon Mechanical Turk). [3]Quantitative evaluation in simulation, PoC on robot. [4]Quantitative on AI subcomponents of the system and then on test set by using robot. [5]PoC on robot + quantitative evaluation of perception subsystem. [6]Quantitative evaluation on dataset obtained from simulation, PoC on robot. [7]One robotic task with a statistical evaluation and another for PoC. [8]Always grasping the drill by the handle. [9]Always grasping for bin picking. [10]Based on manually (Amazon Mechanical Turk) labeled dataset and WordNet, Flickr image co-occurrence statistics. [11]Based on manually labeled dataset. [12]Spatial preposition labeling collected via Amazon Mechanical Turk using 3D virtual environment datasets. [13]Domain-specific ontologies, general ontologies, object models, environment maps, Prolog-based structures. [14]Manually created dataset: natural language commands mapped into "persons behavior when performing those commands in virtual environment", object co-occurrence statistics from image database. [15]Wordnet, Framenet, Wikihow. [16]22 videos × 45 subjects × (on average) 13 commands, for training of the model around 1000 phrases were found to be required. [17]Skill set defined based on 566 industrial tasks. [18]60 cooking recipes with formally specified ingredients and manually annotated robot instructions for each recipe. [19]290 utterances for training (210 for testing), 4000 labeled images for object classification (80 per class). [20]Large amount of different types of knowledge, hard to define numerically. [21]50 natural language commands × one or more scenarios and some compound natural language commands. [22]One demonstration for each task. [23]14 skills are ready for transfer to other robots. [24]300 different cooking task executions in simulation. [25]8 action sequences and CNN learning set derived from those sequences. [26]50,000 pushing attempts. [27]2,898,410 images, 800,000 grasp attempts. [28]58 executions. [29]29 ADTs. [30]Forklift operation. [31]One cooking sequence: pouring several components, mixing, and scraping for emptying. [32]Manipulations based on spatial relations between objects. [33]Demonstration example is very specific (drink-serving in hospital room). [34]Navigation domain. [35]Neutralization and pipette usage. [36]Five cooking tasks. [37]Power drill handling. [38]Pushing. [39]Bin picking. [40]Navigating toward fridge and opening door with hinged joint. [41]Grasping, mounting, moving. [42]Move, lift and varieties (go back, pick up, pull parallel). [43]Pick up and empty (the container). [44]3 tasks on making different dishes, 9 lower-level actions: locating, grasping, lifting, moving, pouring, shaking, mixing, scraping, baking. [45]Pick up, point to, point at, place at, move to. [46]Serving drink to patient. [47]Take, insert, put, calibrate. [48]Bring, meet, avoid, follow, go. [49]Move, glue, grip. [50]Neutralization and pipetting. [51]Put in box and stack, open and close drawer, push, grasp. [52]Inserting switch in box. [53]Number of actions unclear. Tasks are: assembly of gearbox and picking and placing wooden panels. [54]Three tasks of varying complexity: cloning of state chart group for placing objects using another robot; transferring waving motion to another robot; grasping of unknown object using visual servoing. [55]5 cooking tasks; 13 low-level robot commands. [56]Drag, turn, grasp. [57]Grasping. [58]Rotate, place, insert, put over, lay, shake, unscrew, drop, push away, invert. [59]Trucks, trailers, and pallets. [60]Box and feeder. [61]Only bowls. [62]Bed, cabinet, drink bottle. [63]Printed circuit board, fixture, shield. [64]Chemical lab equipment (holders, tubes, pipettes). [65]Blocks and paper cups, drawer, bottle: 3–5 different scenarios for each experiment. [66]Switch and box. [67]Fridge door opening in two different labs. [68]Missing skills in known assembly sequences will be suggested. [69]Many (general) Prolog rules. [70]Only rudimentary mapping objects by names. [71]Rule systems employed, but for action parametrization, not for reuse. [72]Heuristic energy function used. [73]Knowledge adaptation rules. [74]Only at coding level. [75]Through additional read-out procedures.

comfort factors in working with the system will have to be investigated and evaluated in future work.

In summary, we believe that this study is one of the first attempts to provide a more rigorously structured action scaffold for action-code reuse. It rests on intrinsic action properties (the SEC events) not needing potentially arbitrary human definitions. This might make such a structure more “universally agreeable”. Thus, we see the continuation of structuring efforts of action components as one of the most important future efforts to arrive at transferable and reusable robotics code.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This work was supported by the European Community’s Seventh Framework Programme FP7 (Programme and Theme: ICT-2011. 2.1, Cognitive Systems and Robotics) (grant number 600578, ACAT).

Notes

1. http://universaldependencies.org/u/dep/index.html
2. Neighborhood defines the temporal context of the movement primitive within the data structure used for action description; however, we avoided using the word “context” here to avoid mismatch with text-based analysis.
3. https://alexandria.physik3.uni-goettingen.de/cns-group/datasets/cut/
4. https://alexandria.physik3.uni-goettingen.de/cns-group/datasets/cut/

ORCID iD

Jan Matthias Braun https://orcid.org/0000-0003-2749-9000

Supplemental Material

Supplemental material for this article is available online.

References

Aein MJ (2016) *Development and analysis of a library of actions for robot arm-hand systems*. PhD Thesis, Georg-August-Universität Göttingen, Germany.

Aein M, Aksoy E, Tamosiunaite M, et al. (2013) Toward a library of manipulation actions based on semantic object-action relations. In: *2013 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, Tokyo, Japan, 3–7 November 2013, pp. 4555–4562. Piscataway, NJ: IEEE.

Aksoy EE, Abramov A, Dörr J, et al. (2011) Learning the semantics of object-action relations by observation. *The International Journal of Robotics Research* 30(10): 1229–1249.

Aksoy EE, Orhan A and Wörgötter F (2017) Semantic decomposition and recognition of long and complex manipulation action sequences. *International Journal of Computer Vision* 122(1): 84–115.

Aksoy EE, Zhou Y, Wächter M, et al. (2016) Enriched manipulation action semantics for robot execution of time constrained tasks. In: *2016 IEEE-RAS 16th international conference on humanoid robots (humanoids)*, Cancun, Mexico, 15–17 November 2016, pp. 109–116. Piscataway, NJ: IEEE.

Alexandrova S, Tatlock Z and Cakmak M (2015) RoboFlow: A flow-based visual programming language for mobile manipulation tasks. In: *2015 IEEE international conference on robotics and automation (ICRA)*, Seattle, WA, USA, 26–30 May 2015, pp. 5537–5544. Piscataway, NJ: IEEE.

Alterovitz R, Koenig S and Likhachev M (2016) Robot planning in the real world: Research challenges and opportunities. *AI Magazine* 37(2): 76–84.

Angerer A, Hoffmann A, Ortmeier F, et al. (2009) Object-centric programming: A new modeling paradigm for robotic applications. In: *2009 IEEE international conference on automation and logistics*, Shenyang, China, 5–7 August 2009, pp. 18–23. Piscataway, NJ: IEEE.

Artzi Y and Zettlemoyer L (2013) Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the Association for Computational Linguistics* 1(1): 49–62.

Beetz M, Bessler D, Winkler J, et al. (2016) Open robotics research using web-based knowledge services. In: *2016 IEEE international conference on robotics and automation (ICRA)*, Stockholm, Sweden, 16–21 May 2016, pp. 5380–5387. Piscataway, NJ: IEEE.

Billard A, Calinon S, Dillmann R, et al. (2008) Robot programming by demonstration. In: B Siciliano and O Khatib (eds.) *Springer Handbook of Robotics*. Berlin: Springer, pp. 1371–1394.

Björkelund A, Malec J, Nilsson K, et al. (2011) Knowledge and skill representations for robotized production. *IFAC Proceedings Volumes* 44(1): 8999–9004.

Bøgh S, Nielsen OS, Pedersen MR, et al. (2012) Does your robot have skills? In: *43rd international symposium on robotics (ISR)*, Taipei, Taiwan, 29–31 August 2012, pp. 6–12. Berlin: VDE Verlag GMBH.

Bohg J, Morales A, Asfour T, et al. (2014) Data-driven grasp synthesis—a survey. *IEEE Transactions on Robotics* 30(2): 289–309.

Bollini M, Tellex S, Thompson T, et al. (2013) Interpreting and executing recipes with a cooking robot. In: JP Desai, G Dudek, O Khatib, et al. (eds.) *Experimental Robotics* Heidelberg: Springer International Publishing, pp. 481–495.

Bozcuoğlu AK, Kazhoyan G, Furuta Y, et al. (2018) The exchange of knowledge using cloud robotics. *IEEE Robotics and Automation Letters* 3(2): 1072–1079.

Buchin M, Driemel A, Van Kreveld M, et al. (2011) Segmenting trajectories: A framework and algorithms using spatiotemporal criteria. *Journal of Spatial Information Science* 2011(3): 33–63.

Chambers N, Cer D, Grenager T, et al. (2007) Learning alignments and leveraging natural logic. In: *ACL-PASCAL workshop on textual entailment and paraphrasing, RTE ’07*, Prague, Czech Republic, –28–29 June 2007, pp. 165–170. Stroudsburg, PA: Association for Computational Linguistics.

de Marneffe MC and Manning CD (2008) The Stanford typed dependencies representation. In: *Coling 2008: Proceedings of the workshop on cross-framework and cross-domain parser evaluation, CrossParser ’08*, Manchester, UK, —23 August 2008. Stroudsburg, PA: Association for Computational Linguistics.

de Marneffe MC, Dozat T, Silveira N, et al. (2014) Universal Stanford dependencies: A cross-linguistic typology. In: *Ninth international conference on language resources and evaluation (LREC)* (eds. N Calzolari, K Choukri, T Declerck, et al.), Reykjavik, Iceland, 26–31 May 2014, pp. 4585–4592. Paris: European Language Resources Association.

Dillmann R (2004) Teaching and learning of robot tasks via observation of human performance. *Robotics and Autonomous Systems* 47(2–3): 109–116.

Drath R, Lüder A, Peschke J, et al. (2008) AutomationML—the glue for seamless automation engineering. In: *International conference on emerging technologies and factory automation (ETFA)*, Hamburg, Germany, 5–18 September 2008, pp. 616–623. Piscataway, NJ: IEEE.

Erdem E, Patoglu V and Saribatur ZG (2015) Integrating hybrid diagnostic reasoning in plan execution monitoring for cognitive factories with multiple robots. In: *2015 IEEE international conference on robotics and automation (ICRA)*, Seattle, WA, USA, 26–30 May 2015, pp. 2007–2013. Piscataway, NJ: IEEE.

Finn C and Levine S (2017) Deep visual foresight for planning robot motion. In: *IEEE international conference on robotics and automation (ICRA)*, Singapore, 29 May–3 June 2017, pp. 2786–2793. Piscataway, NJ: IEEE.

Fischer K, Kirstein F, Jensen LC, et al. (2016) A comparison of types of robot control for programming by demonstration. In: *2016 11th ACM/IEEE international conference on human–robot interaction (HRI)*, Christchurch, New Zealand, 7–10 March 2016, pp. 213–220. Piscataway, NJ: IEEE.

Gaspar T, Ridge B, Bevec R, et al. (2017) Rapid hardware and software reconfiguration in a robotic workcell. In: *18th international conference on advanced robotics (ICAR)*, Hong Kong, China, 10–12 July 2017, pp. 229–236. Piscataway, NJ: IEEE.

Gemignani G, Bastianelli E and Nardi D (2015) Teaching robots parametrized executable plans through spoken interaction. In: *2015 International conference on autonomous agents and multiagent systems, AAMAS ’15*, Istanbul, Turkey, 4–8 May 2015, pp. 851–859. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.

Guadarrama S, Riano L, Golland D, et al. (2013) Grounding spatial relations for human–robot interaction. In: *2013 IEEE/RSJ international conference on intelligent robots and systems*, Tokyo, Japan, 3–7 November 2013, pp. 1640–1647. Piscataway, NJ: IEEE.

Haidu A and Beetz M (2016) Action recognition and interpretation from virtual demonstrations. In: *International conference on intelligent robots and systems (IROS)*. Daejeon, South Korea, 9–14 October 2016, pp. 2833–2838. Piscataway, NJ: IEEE.

Hart S, Dinh P and Hambuchen K (2015) The affordance template ROS package for robot task programming. In: *2015 IEEE international conference on robotics and automation (ICRA)*, Seattle, WA, USA, 26–30 May 2015, pp. 6227–6234. Piscataway, NJ: IEEE.

Huang DW, Katz GE, Langsfeld JD, et al. (2015) An object-centric paradigm for robot programming by demonstration. In: Fiopiastis CM and Schmorrow DD (eds.) *Foundations of Augmented Cognition*. Cham: Springer International Publishing, pp. 745–756.

Huang J, Lau T and Cakmak M (2016) Design and evaluation of a rapid programming system for service robots. In: *Eleventh ACM/IEEE international conference on human robot interaction, HRI ’16*, Christchurch, New Zealand, 7–10 March 2016, pp. 295–302. Piscataway, NJ: IEEE.

Kipper K, Korhonen A, Ryant N, et al. (2006) Extending VerbNet with novel verb classes. In: *Fifth international conference on language resources and evaluation (LREC-2006)*, Genoa, Italy, 22–28 May 2006. Paris: European Language Resources Association (ELRA).

Klein D and Manning CD (2004) Corpus-based induction of syntactic structure: Models of dependency and constituency. In: *42nd annual meeting of Association for Computational Linguistics, ACL ’04*, Barcelona, Spain, 21–26 July 2004, pp. 478–486. Stroudsburg, PA, USA: Association for Computational Linguistics.

Kober J, Bagnell JA and Peters J (2013) Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research* 32(11): 1238–1274.

Kollar T, Tellex S, Roy D, et al. (2014) Grounding verbs of motion in natural language commands to robots. In: O Khatib, V Kumar and G Sukhatme (eds.) *Experimental Robotics*. Berlin: Springer, pp. 31–47.

Kong W and Ranganath S (2008) Automatic hand trajectory segmentation and phoneme transcription for sign language. In: *8th IEEE international conference on automatic face & gesture recognition, FG’08*, Amsterdam, Netherlands, 17–19 September 2008. Piscataway, NJ: IEEE.

Kramberger A, Gams A, Nemec B, et al. (2016) Transfer of contact skills to new environmental conditions. In: *2016 IEEE-RAS 16th international conference on humanoid robots (humanoids)*, Cancun, Mexico, 15–17 November 2016, pp. 668–675. Piscataway, NJ: IEEE.

Ku LY, Learned-Miller E and Grupen R (2017a) An aspect representation for object manipulation based on convolutional neural networks. In: *2017 IEEE international conference on robotics and automation (ICRA)*, Singapore, 29 May–3 June 2017, pp. 794–800. Piscataway, NJ: IEEE.

Ku LY, Learned-Miller E and Grupen R (2017b) Associating grasp configurations with hierarchical features in convolutional neural networks. In: *IEEE/RSJ international conference on intelligent robots and systems (IROS)*, Vancouver, Canada, 24–28 September 2017, pp. 2434–2441. Piscataway, NJ: IEEE.

Latombe JC (2012) *Robot Motion Planning*. New York, NY: Springer Science & Business Media.

Levine S, Pastor P, Krizhevsky A, et al. (2018) Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research* 37(4–5): 421–436.

Lippiello V, Ruggiero F, Siciliano B, et al. (2013) Visual grasp planning for unknown objects using a multifingered robotic hand. *IEEE/ASME Transactions on Mechatronics* 18(3): 1050–1059.

Lisca G, Nyga D, Bálint-Benczédi F, et al. (2015) Towards robots conducting chemical experiments. In: *2015 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, Hamburg, Germany, 28 September–2 October 2015, pp. 5202–5208. Piscataway, NJ: IEEE.

Macfarlane S and Croft EA (2003) Jerk-bounded manipulator trajectory planning: Design for real-time applications. *IEEE Transactions on Robotics and Automation* 19(1): 42–52.

Manschitz S, Kober J, Gienger M, et al. (2014) Learning to sequence movement primitives from demonstrations. In: *2014 IEEE/RSJ international conference on intelligent robots and systems*, Chicago, IL, USA, 14–18 September 2014, pp. 4414–4421. Piscataway, NJ: IEEE.

Markievicz I, Vitkute-Adzgauskiene D and Tamosiunaite M (2013) Semi-supervised learning of action ontology from domain-specific corpora. In: T Skersys, R Butleris and R Butkiene (eds.) *Information and Software Technologies*. Berlin: Springer, pp. 173–185.

Matuszek C, Herbst E, Zettlemoyer L, et al. (2013) Learning to parse natural language commands to a robot control system. In: J Desai, G Dudek, O Khatib, et al. (eds.) *Experimental Robotics*. Heidelberg: Springer International Publishing, pp. 403–415.

Miller E (1998) An introduction to the resource description framework. *Bulletin of the American Society for Information Science and Technology* 25(1): 15–19.

Miller GA, Beckwith R, Fellbaum C, et al. (1990) Wordnet: An on-line lexical database. *International Journal of Lexicography* 3: 235–244.

Misra DK, Sung J, Lee K, et al. (2016) Tell me Dave: Context-sensitive grounding of natural language to manipulation instructions. *The International Journal of Robotics Research* 35(1–3): 281–300.

Moradi Dalvand M and Nahavandi S (2014) Teleoperation of ABB industrial robots. *Industrial Robot: An International Journal* 41(3): 286–295.

Nakamura A, Nagata K, Harada K, et al. (2013) Error recovery using task stratification and error classification for manipulation robots in various fields. In: *2013 IEEE/RSJ international conference on intelligent robots and systems*, Tokyo, Japan, 3–7 November, 2013, pp. 3535–3542. Piscataway, NJ: IEEE.

Niemueller T, Lakemeyer G and Srinivasa SS (2012) A generic robot database and its application in fault analysis and performance evaluation. In: *2012 IEEE/RSJ international conference on intelligent robots and systems*, Vilamoura, Portugal, 7–12 October 2012, pp. 364–369. Piscataway, NJ: IEEE.

Nivre J and Nilsson J (2005) Pseudo-projective dependency parsing. In: *43rd annual meeting of the Association for Computational Linguistics, ACL '05*, Ann Arbor, MI, 25–30 June 2005, pp. 99–106. Stroudsburg, PA: Association for Computational Linguistics.

Nyga D and Beetz M (2012) Everything robots always wanted to know about housework (but were afraid to ask). In: *2012 IEEE/RSJ international conference on intelligent robots and systems*, Vilamoura, Portugal, 7–12 October 2012, pp. 243–250. Piscataway, NJ: IEEE.

Ovchinnikova E, Wächter M, Wittenbeck V, et al. (2015) Multipurpose natural language understanding linked to sensorimotor experience in humanoid robots. In: *2015 IEEE-RAS 15th international conference on humanoid robots (humanoids)*, Seoul, South Korea, 3–5 November 2015, pp. 365–372. Piscataway, NJ: IEEE.

Persson J, Gallois A, Björkelund A, et al. (2010) A knowledge integration framework for robotics. In: *ISR 2010 (41st international symposium on robotics) and ROBOTIK 2010 (6th German conference on robotics)*, Munich, Germany, 7–9 June 2010. Piscataway, NJ: IEEE.

Perzylo A, Griffiths S, Lafrenz R, et al. (2015) Generating grammars for natural language understanding from knowledge about actions and objects. In: *2015 IEEE international conference on robotics and biomimetics (ROBIO)*, Zhuhai, China, 6–9 December 2015, pp. 2008–2013. Piscataway, NJ: IEEE.

Perzylo A, Somani N, Profanter S, et al. (2016) Intuitive instruction of industrial robots: Semantic process descriptions for small lot production. In: *2016 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, Daejeon, South Korea, 9–14 October 2016, pp. 2293–2300. Piscataway, NJ: IEEE.

Profanter S, Perzylo A, Somani N, et al. (2015) Analysis and semantic modeling of modality preferences in industrial human–robot interaction. In: *2015 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, Hamburg, Germany, 28 September–2 October 2015, pp. 1812–1818. Piscataway, NJ: IEEE.

Riazuelo L, Tenorth M, Marco DD, et al. (2015) RoboEarth semantic mapping: A cloud enabled knowledge-based approach. *IEEE Transactions on Automation Science and Engineering* 12(2): 432–443.

Rosenthal S, Selvaraj SP and Veloso M (2016) Verbalization: Narration of autonomous mobile robot experience. In: *IJCAI'16, the 26th international joint conference on artificial intelligence*. New York City, NY, USA, 9–15 June 2016, pp. 862–868. Menlo Park, CA: AAAI Press.

Schlette C, Losch D and Rossmann J (2014) A visual programming framework for complex robotic systems in micro-optical assembly. In: *ISR/Robotik 2014; 41st international symposium on robotics*, Munich, Germany, 2–3 June 2014. Piscataway, NJ: IEEE.

Schou C, Damgaard JS, Bøgh S, et al. (2013) Human–robot interface for instructing industrial tasks using kinesthetic teaching. In: *IEEE ISR 2013*, Seoul, South Korea, 24–26 October 2013. Piscataway, NJ: IEEE.

Shadbolt N, Berners-Lee T and Hall W (2006) The Semantic Web revisited. *IEEE Intelligent Systems* 21(3): 96–101.

She L, Yang S, Cheng Y, et al. (2014) Back to the blocks world: Learning new actions through situated human–robot dialogue. In: *15th annual meeting of the special interest group on discourse and dialogue (SIGDIAL)*, Philadelphia, PA, USA, 18–20 June 2014, pp. 89–97. Stroudsburg, PA: Association for Computational Linguistics.

Stampfer D and Schlegel C (2014) Dynamic state charts: Composition and coordination of complex robot behavior and reuse of action plots. *Intelligent Service Robotics* 7(2): 53–65.

Steinmetz F and Weitschat R (2016) Skill parametrization approaches and skill architecture for human–robot interaction. In: *2016 IEEE international conference on automation science and engineering (CASE)*, Fort Worth, TX, USA, 21–25 August 2016, pp. 280–285. Piscataway, NJ: IEEE.

Stenmark M and Nugues P (2013) Natural language programming of industrial robots. In: *44th international symposium on robotics (ISR)*, Seoul, South Korea, 24–26 October 2013. Piscataway, NJ: IEEE.

Stenmark M, Malec J and Stolt A (2015) From high-level task descriptions to executable robot code. In: Filev D, Jablkowski J, Kacprzyk J, et al. (eds.) *Intelligent Systems '2014: Advances in Intelligent Systems and Computing*. Cham: Springer, pp. 189–202.

Stulp F, Fedrizzi A, Mösenlechner L, et al. (2012) Learning and reasoning with action-related places for robust mobile manipulation. *Journal of Artificial Intelligence Research* 43: 1–42.

Sucan IA, Moll M and Kavraki LE (2012) The open motion planning library. *IEEE Robotics Automation Magazine* 19(4): 72–82.

Tellex S, Kollar T, Dickerson S, et al. (2011) Understanding natural language commands for robotic navigation and mobile manipulation. In: *Twenty-fifth AAAI conference on artificial intelligence (AAAI)*. San Francisco, CA, USA, 7–11 August 2011, pp. 1507–1514. Menlo Park, CA: AAAI Press.

Tenorth M and Beetz M (2013) KnowRob: A knowledge processing infrastructure for cognition-enabled robots. *The International Journal of Robotics Research* 32(5): 566–590.

Tenorth M, Perzylo AC, Lafrenz R, et al. (2013) Representation and exchange of knowledge about actions, objects, and environments in the RoboEarth framework. *IEEE Transactions on Automation Science and Engineering* 10(3): 643–651.

Thomason J, Zhang S, Mooney R, et al. (2015) Learning to interpret natural language commands through human–robot dialog. In: *2015 international joint conference on artificial intelligence (IJCAI)*, Buenos Aires, Argentina, 25–31 July 2015, pp. 1923–1929. Menlo Park, CA: AAAI Press.

Vahrenkamp N, Asfour T and Dillmann R (2012) Simultaneous grasp and motion planning: Humanoid robot ARMAR-III. *IEEE Robotics Automation Magazine* 19(2): 43–57.

Vezzani G, Pattacini U and Natale L (2017) A grasping approach based on superquadric models. In: *IEEE international conference on robotics and automation (ICRA)*, Singapore, 29 May–3 June 2017, pp. 1579–1586. Piscataway, NJ: IEEE.

Wächter M, Ottenhaus S, Kröhnert M, et al. (2016) The ArmarX statechart concept: Graphical programming of robot behavior. *Frontiers in Robotics and AI* 3: 33.

Waibel M, Beetz M, Civera J, et al. (2011) RoboEarth. *IEEE Robotics Automation Magazine* 18(2): 69–82.

Walter MR, Hemachandra S, Homberg B, et al. (2014) A framework for learning semantic maps from grounded natural language descriptions. *The International Journal of Robotics Research* 31(9): 1167–1190.

Williams T, Johnson C, Scheutz M, et al. (2017) A tale of two architectures: A dual-citizenship integration of natural language and the cognitive map. In: *16th conference on autonomous agents and multi-agent systems* (eds. S Das, E Durfee, K Larson, et al.), Sao Paolo, Brazil, 8–12 May 2017, pp. 1360–1368. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems.

Winkler J, Tenorth M, Bozcuoglu AK, et al. (2014) CRAMm—memories for robots performing everyday manipulation activities. *Advances in Cognitive Systems* 3: 47–66.

Wörgötter F, Aksoy EE, Krüger N, et al. (2013) A simple ontology of manipulation actions based on hand-object relations. *IEEE Transactions on Autonomous Mental Development* 05(2): 117–134.

Appendix: Additional algorithmic details

Here, we explain a few remaining algorithmic details to make our algorithm fully reproducible. These are the EXTRACT_NEIGHBORHOOD and COMPARE_NEIGHBORHOODS procedures used in the action-template-based analysis, as well as the procedure TRANSFORM_COORDINATES used for adaptation of movement primitives taken from existing ADTs to a new situation (ADT cut & recombine phase). We also define the similarity measures used for hybrid similarity evaluation and explain how we acquire scene data; this is, however, independent of the rest of the algorithmic procedures on ADT data reuse and can be chosen freely.

Neighborhood extraction

The inputs to the procedure EXTRACT_NEIGHBORHOOD are:

- action name;
- action template;
- index pair (i, j), where i denotes the number of the SEC state (column) and j is the movement primitive number in that column.

The procedure outputs the neighborhood elements as indicated in Algorithm 2.

Algorithm 2. Procedure for specifying neighborhood.

```
1: procedure EXTRACT_NEIGHBORHOOD
2:     Extract all object pairs for which SEC relations are indicated
       from the action template "action name".
3:     Extract SEC columns $i$ and $i+1$ from the action template.
4:     Extract movement primitives $\max(1, j-1)$ to
       $\min(\#primitives,\ j+1)$ from the column $i$ in the action
       template.
5:     Define elements extracted in 2 to 4 as the neighborhood $e$
       for movement primitive $(i,j)$.
```

Neighborhood comparison

The procedure COMPARE_NEIGHBORHOODS (see Algorithm 3) indicates how we compare neighborhoods e_1 and e_2 of two movement primitives. The output is binary: "matching" or "non-matching".

Algorithm 3. Procedure for comparing neighborhoods e_1 and e_2.

```
1: procedure COMPARE_NEIGHBORHOODS
2:     Initialize the result to "non-matching".
3:     Match object pairs for which SEC relations are calculated
       in neighborhoods $e_1$ and $e_2$ and divide those into two subsets:
       "matching pairs" and "non-matching pairs".
4:     if SEC entries for the "matching pairs" match in $e_1$ and $e_2$ then
5:         if the SEC entries in the "non-matching pairs" always
           remain in the state $N$ (not touching) then
6:             if the movement primitive names and object
               denominators match in $e_1$ and $e_2$ then
7:                 Set the result to "matching".
```

Scene interpretation and coordinate transformation

Scene data were acquired by placing objects in canonical poses and kinematically tracking them throughout the movement. We considered our objects as center symmetric, which was perfectly true for bottles, jars, lids, bowls, and measuring beakers, but was an approximation for boxes, trays, spoons and cups (the latter, owing to the cup handle).

The procedure TRANSFORM_COORDINATES (see Algorithm 4) was used to adapt the TCP end pose in the *arm_move* movement primitives extracted from existing ADTs to the new coordinates in the scene. We describe the horizontal (table) plane as (X, Y) and the vertical direction

Algorithm 4. Transforming goal point of the movement primitive *arm_move* for object denominator *main*.

```
procedure TRANSFORM_COORDINATES
    Determine in the ADT the center of the TCP at the start of the movement primitive: $(x1^{\mathrm{s}}_{\mathrm{TCP}}, y1^{\mathrm{s}}_{\mathrm{TCP}}, z1^{\mathrm{s}}_{\mathrm{TCP}})$.
    Determine in the ADT the center of the main object: $(x1_{\mathrm{m}}, y1_{\mathrm{m}}, z1_{\mathrm{m}})$.
    Determine in the ADT the TCP at the end of the movement primitive: $(x1^{\mathrm{e}}_{\mathrm{TCP}}, y1^{\mathrm{e}}_{\mathrm{TCP}}, z1^{\mathrm{e}}_{\mathrm{TCP}})$.
    Determine in the real scene the center of the main object: $(x2_{\mathrm{m}}, y2_{\mathrm{m}}, z2_{\mathrm{m}})$.
    Determine the actual start TCP in the real scene: $(x2^{\mathrm{s}}_{\mathrm{TCP}}, y2^{\mathrm{s}}_{\mathrm{TCP}}, z2^{\mathrm{s}}_{\mathrm{TCP}})$.
    if $\mathrm{abs}((x1^{\mathrm{s}}_{\mathrm{TCP}}, y1^{\mathrm{s}}_{\mathrm{TCP}})' - (x1_{\mathrm{m}}, y1_{\mathrm{m}})') > \epsilon$ and $\mathrm{abs}((x2^{\mathrm{s}}_{\mathrm{TCP}}, y2^{\mathrm{s}}_{\mathrm{TCP}})' - (x2_{\mathrm{m}}, y2_{\mathrm{m}}))' > \epsilon$ then
        Determine the coordinate frame of the main object in the $(X, Y)$ plane in the ADT based on the vector: $\vec{F}_{\mathrm{ADT}} = (x1^{\mathrm{s}}_{\mathrm{TCP}}, y1^{\mathrm{s}}_{\mathrm{TCP}})' - (x1_{\mathrm{m}}, y1_{\mathrm{m}})'$.
        Determine the endpoint offset in the ADT as vector: $\vec{o}_{\mathrm{ADT}} = (x1^{\mathrm{e}}_{\mathrm{TCP}}, y1^{\mathrm{e}}_{\mathrm{TCP}})' - (x1_{\mathrm{m}}, y1_{\mathrm{m}})'$.
        Determine the coordinate frame of the main object in $(X, Y)$ plane in the real scene based on the vector: $\vec{F}_{\mathrm{real}} = (x2^{\mathrm{s}}_{\mathrm{TCP}}, y2^{\mathrm{s}}_{\mathrm{TCP}})' - (x2_{\mathrm{m}}, y2_{\mathrm{m}})'$.
        Transform $\vec{o}_{\mathrm{ADT}}$ from frame $\vec{F}_{\mathrm{ADT}}$ into $\vec{F}_{\mathrm{real}}$ and obtain $\vec{o}_{\mathrm{real}}$.
        Define the endpoint of the arm_move primitive in the $(X, Y)$ plane as $(x2_{\mathrm{m}}, y2_{\mathrm{m}})' + \vec{o}_{\mathrm{real}}$.
    else
        Take care of singular cases.
    Define the offset in $Z$ direction in the ADT: $o_z = z1^{\mathrm{e}}_{\mathrm{TCP}} - z1_{\mathrm{m}}$.
    Define the endpoint of the arm_move primitive in the $Z$ direction as: $z2_{\mathrm{m}} + o_z$.
```

as Z. As the objects used in our experiments were (or were approximated as) axially symmetric, we used the acquired degree of freedom for orienting the object frames based on the vector between the TCP and the main object center in the (X, Y) plane measured at the start point of the movement. This was done in the case of the object denominator *main*, where for the denominators *primary* and *secondary* the object frames were based on relations between the center of the main and primary or secondary objects, respectively. This allowed the ADT-defined relative positions oriented with the direction of the movement in the (X, Y) plane to be preserved. The Z (vertical) direction was treated separately based on the relative position with respect to object center points in Z. The (X, Y) plane and Z direction were treated separately, as the scene configuration in this and other table-top scenes is normally expressed in the (X, Y) plane, whereas the Z direction shows only the height of the objects.

The procedure TRANSFORM_COORDINATES (see Algorithm 4) only shows how the goal position of the *arm_move* is acquired in the case of object denominator *main*. For the object denominators *primary* and *secondary*, the main object position is extracted instead of the TCP and the primary or secondary object position is extracted instead of the main object position.

One can derive the orientation of the TCP at the end of the *arm_move* primitive in a similar way, reusing the relative orientations in (X, Y) and absolute orientations in Z to (X, Y) from the ADT. However, we implemented orientation in this way only for the *place* action we performed in our experiments, while in the remaining actions, we only took the initial orientation of the TCP with respect to the main object from the ADT, and kept it throughout the rest of the action to reduce the complexity of the implementation.

Table 12. Weights used in equation 2 for different object denominators.

	Case 1 Obj. denom. *main*	Case 2 Obj. denom. *primary*	Case 3 Obj. denom. *secondary*
ω_1	8	8	8
ω_2	10	5	5
ω_3	1	10	1
ω_4	1	1	10

Measures for hybrid similarity evaluation

To combine symbolic and sub-symbolic components, we re-implemented the symbolic ranking given in Table 7 on the basis of a weighting procedure

$$S_{\mathrm{symb}} = (\omega_1 \delta(a_{\mathrm{c}}, a_{\mathrm{ADT}}) + \omega_2 \delta(main_{\mathrm{c}}, main_{\mathrm{ADT}}) + \omega_3 \delta(prim_{\mathrm{c}}, prim_{\mathrm{ADT}}) + \omega_4 \delta(sec_{\mathrm{c}}, sec_{\mathrm{ADT}})) / \sum_{i=1}^{4} \omega_i, \tag{2}$$

where a_{c} and a_{ADT} are action names in the new command and the ADT, $main_{\mathrm{c}}$, $prim_{\mathrm{c}}$, and sec_{c} are main, primary, and secondary object names in the command, while the $main_{\mathrm{ADT}}$, $prim_{\mathrm{ADT}}$ and sec_{ADT} are the corresponding object names in the ADT; δ is the Kronecker delta indicating the equality of two names n_1 and n_2

$$\delta(n_1, n_2) = \begin{cases} 1 & \text{if } n_1 = n_2 \\ 0 & \text{if } n_1 \neq n_2 \end{cases}$$

Weights $\omega_1, \ldots, \omega_4$ depend on the object denominator and are given in Table 12. We have chosen the weights so that the resulting similarity S_{symb} reproduces the ranking given in Table 7.

To evaluate object bounding box similarity at the sub-symbolic level, we used the intersection over union (IoU) measure

$$S_{\text{box}} = \frac{volume(B_{\text{scene}} \cap B_{\text{ADT}})}{volume(B_{\text{scene}} \cup B_{\text{ADT}})}$$

where B_{scene} and B_{ADT} are the axis-aligned bounding boxes of the corresponding objects in the scene and in the ADT. The axes are defined as follows: we describe the horizontal (table) plane as (X, Y) and the vertical direction as Z. We measure objects in the canonical poses, where X is the smaller horizontal dimension and Y is the larger horizontal dimension of an object.

Both measures S_{symb} and S_{box} are ranged in the interval $[0, 1]$ and we combine the two as indicated in the main text in Subsection 5.2.2.