Structural bootstrapping - A novel, generative mechanism for faster and more efficient acquisition of action-knowledge

Florentin Wörgötter[a,i], Chris Geib[b,c,i], Minija Tamosiunaite[a,d,i], Eren Erdal Aksoy[a], Justus Piater[e], Hanchen Xiong[e], Ales Ude[f], Bojan Nemec[f], Dirk Kraft[g], Norbert Krüger[g], Mirko Wächter[h], Tamim Asfour[h]

[a]*Georg-August-Universität Göttingen, Bernstein Center for Computational Neuroscience, Department for Computational Neuroscience, III Physikalisches Institut - Biophysik, Göttingen, Germany*
[b]*School of Informatics, Edinburgh, United Kingdom*
[c]*College of Computing and Informatics, Drexel University, Philadelphia, USA*
[d]*Department of Informatics, Vytautas Magnus University, Kaunas, Lithuania*
[e]*Institute of Computer Science, University of Innsbruck, Innsbruck, Austria*
[f]*Humanoid and Cognitive Robotics Lab, Dept. of Automatics, Biocybernetics, and Robotics, Jožef Stefan Institute, Ljubljana, Slovenia*
[g]*Cognitive and Applied Robotics Group, University of Southern Denmark, Odense, Denmark*
[h]*Institute for Anthropomatics and Robotics, Karlsruhe Institute of Technology, Karlsruhe, Germany*
[i]*These authors have contributed equally to this work.*

Abstract

Humans, but also robots, learn to improve their behavior. Without existing knowledge, learning either needs to be explorative and, thus, slow or – to be more efficient – it needs to rely on supervision, which may not always be available. However, once some knowledge base exists an agent can make use of it to improve learning efficiency and speed. This happens for our children at the age of around three when they very quickly begin to assimilate new information by making guided guesses how this fits to their prior knowledge. This is a very efficient *generative learning mechanism* in the sense that the existing knowledge is generalized into as-yet unexplored, novel domains. So far generative learning has not been employed for robots and robot learning remains to be a slow and tedious process. The goal of the current study is to devise for the first time a general framework for a generative process that will improve learning and which can be applied at all different levels of the robot's cognitive architecture. To this end, we introduce the concept of structural bootstrapping – borrowed and modified from child language acquisition – to define a probabilistic process that uses existing knowledge together with new observations to supplement our robot's data-base with missing information about planning-, object-, as well as action-relevant entities. In a kitchen scenario, we use the example of making batter by pouring and mixing two components and show that the agent can efficiently acquire new knowledge about planning operators, objects as well as required motor pattern for stirring by structural bootstrapping. Some benchmarks are shown, too, that demonstrate how structural bootstrapping improves performance.

Keywords: Generative Model, Knowledge Acquisition, Fast Learning

Introduction

It has been a puzzling question how small children at the age of three to four are suddenly able to very quickly acquire the meaning of more and more words in their native language, while at a younger age language acquisition is much slower. Two interrelated processes are being held responsible for this speeding-up. The primary process is semantic bootstrapping where the child associates meaning from observing their world with co-occurring components of sentences. For example, if the word "fill" is consistently uttered in situations where "filling" occurs, then the meaning of the word can be probabilistically guessed from having observed the corresponding action again and again [1, 2]. Once a certain amount of language has been acquired, a second process – named syntactic bootstrapping – can speed this up even more and this is achieved by exploiting structural similarity between linguistic elements. This process can take place entirely within language and happens in a purely symbolic way without influence from the world. For example, if a child knows the meaning of "fill the cup" and then hears the sentence "fill the bowl", it can infer that a "bowl" denotes a thing that can be filled (rather than a word meaning the same thing as "fill") without ever having seen one ([1, 3, 4, 5, 6, 7, 8, 9] see [10] for a comparison between semantic and syntactic bootstrapping). Thus, the most probable meaning of a new word is being estimated on the basis of the prior probability established by previously encountered words of the same semantic and syntactic type in similar syntactic and semantic contexts.

These two generalization mechanisms – semantic and syntactic bootstrapping – are very powerful and allow young humans to acquire language without explicit instruction. It is arguable that bootstrapping is what fuels the explosion in language and conceptual development that occurs around the third year of child development [8, 11].

In general "the trick" seems to be that the child possesses at this age already enough well-ordered knowledge (grammar, word & world knowledge) which allows him/her to perform guided inference without too many unknowns. Grammar and word-knowledge are highly structured symbolic representations and can, thus, provide a solid scaffold for the bootstrapping of language. Symbolic representations, however, do not stop short at human language. For robots, planning, planning operators, and planning languages constitute another (non-human) symbolic domain with which they need to operate. Thus, it seems relatively straightforward to transfer the idea of semantic and syntactic bootstrapping to the planning domain for robot actions. The current paper will first address this problem.

The question, however, arises whether related mechanisms might also play a role for the acquisition of other, non-linguistic cognitive concepts, for example the properties of objects and tools. Briefly, if you know how to peel a potato with a knife, would there be a way to infer that a potato peeler can be used for the same purpose? This example belongs to the second set of problems addressed in this study: How can a cognitive agent infer role and use of different objects employing the knowledge of previously seen (and used) objects, how can it infer the use of movement and force patterns, etc.?

The goal of the current study is to address one complex scenario all the way from the planning-level down to sub-symbolic sensorimotor levels and implement (different) bootstrapping processes for the fast acquisition of action knowledge. The only requirement for all these different bootstrapping mechanisms is that there exists a well-structured scaffold as a basis from where on different inference processes can take place. The different scaffolds, thus, form the structures upon which bootstrapping can be built. Hence, we call these processes "structural bootstrapping".

One can consider structural bootstrapping as a type of semi-supervised probabilistic learning, where an agent uses an internal model (scaffold) to quickly slot novel information (obtained for example by observing a human) into appropriate model categories. This is a *generative process* because existing knowledge is generalized into novel domains, which so far had not been explored. The advantage of such a bootstrapping process is that the agent will be able to very quickly perform these associations and grounding needs only to take place afterwards by experimenting in a guided way with the new piece of knowledge. Evidently, as this is based on probabilistic guesswork, bootstrapping can also lead to wrong results. Still, if the scaffold is solid enough all this can be expected to be much faster and more efficient than the much more unstructured and slow process of bottom-up exploration learning or than full-fledged learning from demonstration. Thus, structural bootstrapping is a way for the generative acquisition and extension of knowledge by which an agent can more efficient redeploy what it currently knows, but where its existing knowledge cannot be *directly* employed. The distinction between syntactic and semantic components is, however, less evident when considering structural (e.g. sensori-motor) elements. It will become clear by the examples below that structural bootstrapping often contains both aspects.

Here we will show that one can implement structural bootstrapping across different levels of our robotics architecture in the humanoid robot ARMAR-III [12, 13] trying to demonstrate that bootstrapping appears in different guises and will, thus, possibly not be limited to the case studies presented in this paper. As a major aspect, this work is meant to advocate structural bootstrapping as a way forward to a more efficient extension of robot-knowledge in the future. Early on we emphasize that the complexity of the here-shown aspects prevents exhaustive analyses. After all we are dealing with very complicated and possibly human-like cognitive generative (inference) processes for which children and adults need years of experience to reach their final efficiency.

The paper is structured in the following way. First we provide an overview of the bootstrapping idea, then we show details on the system, processes, and methods. Next we show six different types of structural bootstrapping at different levels. This will be followed by some benchmarks and a discussion section which also includes the state of the art in robot knowledge acquisition.

Overview

The goal of this work is to use a humanoid robot (ARMAR III) to demonstrate several ways to perform structural bootstrapping at different levels of its intrinsic cognitive architecture. Thus, we define a traditional 3-layer architecture consisting of a Planning level, a Mid-level, and a Sensorimotor Level [14]. In order to perform a task, the robot first needs to make a (symbolic) plan. The mid-level acts as a symbol-to-signal mediator (explained below) and couples the planning information to the sensorimotor (signal) level. The sensorimotor level then performs execution but also sensing of the situation and the progress and potential errors of the robot's motor actions. Details of the actual sensorimotor control loops shall be omitted here for the sake of brevity (see e.g. [14] for this).

Every layer uses different syntactic elements; for example the Planning layer uses Planning Operators. But all syntactic elements will always be embedded in their layer-specific scaffold. For the Planning layer its is the Planning Language that defines how to arrange and use the Planning Operators. Hence the Planning Language is the scaffold of this layer. Similar structural relations between syntactic elements and scaffolds are defined for the two other layers.

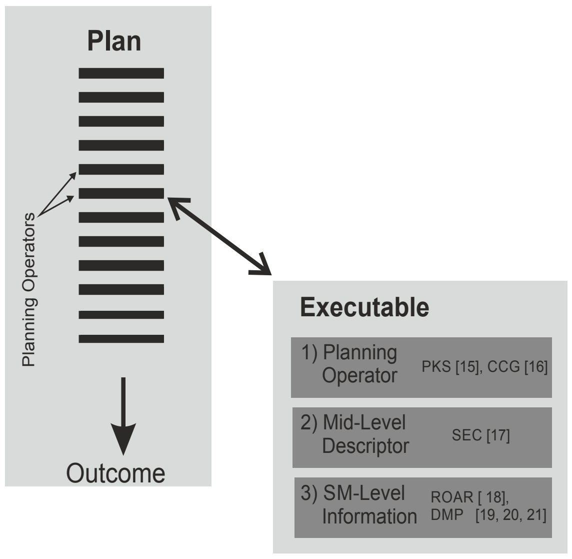


Figure 1: Structure of an Executable and its link to the robotics plan.

The general structural bootstrapping idea is now rather simple: Semantic and/or syntactic similarity at the level of the scaffold is used to infer, which (known) syntactic entities can take the role of which other (unknown, but currently observed) syntactic entities. In other words: Using the appropriate layer-specific scaffold, the robot makes inferences about the role of an observed but "incomprehensible" entity, for which the machine does not have any representation in its knowledge base. Based on these inferences the unknown entity can be replaced with one that is known (one, for which their is an entry existing in the knowledge base). This replacement will allow the machine to continue with its planned operation ideally without any additional information.

Structures

To allow bootstrapping we need to define the actual data structures, which are used by the robot for execution of a task and which need to be set up in a way to allow for structural bootstrapping, too (Fig. 1).

At the top layer we use a conventional robotics planner [15] to create a plan for a given task. The plan consists of a sequence of Planning Operators. As such these planning operators cannot be executed by a robot. Thus, to achieve this, we define a so-called *Executable*, which consists of several components using methods from the literature:

1. a planning operator, by which the Executable is linked to the Plan [15, 16], together with its
2. mid-level descriptors [17] and
3. all perception and/or control information from the sensorimotor level for executing an action [18, 19, 20, 21].

Hence, during execution the different planning operators are called-up and each one – in turn – calls the belonging Executable, which contains the required control information to actually execute this chunk of the plan.

Some of these aspects are to some degree embodiment specific (most notably the control information), some others are not. Note, the structure of an Executable is related to the older concept of an Object-Action-Complex (OAC, [22, 23]). OACs had been defined in our earlier works as rather abstract entities [23], the Executables – as defined – here extend the OAC concept by now also including planning operators and are finally defined in a very concrete way (to actually allow for execution, which had not yet been the case for the OAC).

Essential to this is work is that we use the concept of bootstrapping now in the same way at these three levels. The syntactic representations used to compute aspects of a given level are level-dependent where we have the following syntactic representatives:

1. planning operators,
2. syntactic structure of mid-level descriptors, and
3. perceptual (sensor) and control (motor) variables.

Therefore, we employ different (grammatical) scaffolds for the bootstrapping:

1. planning language,
2. semantic event chains (SECs[1] [24, 17]), and
3. sensorimotor feature/parameter regularity

from where the bootstrapping commences.

[1]Semantic Event Chains (SECs) encode for an action the sequence of touching and untouching events that happen until the action concludes. A more detailed description is given in the Methods section below.

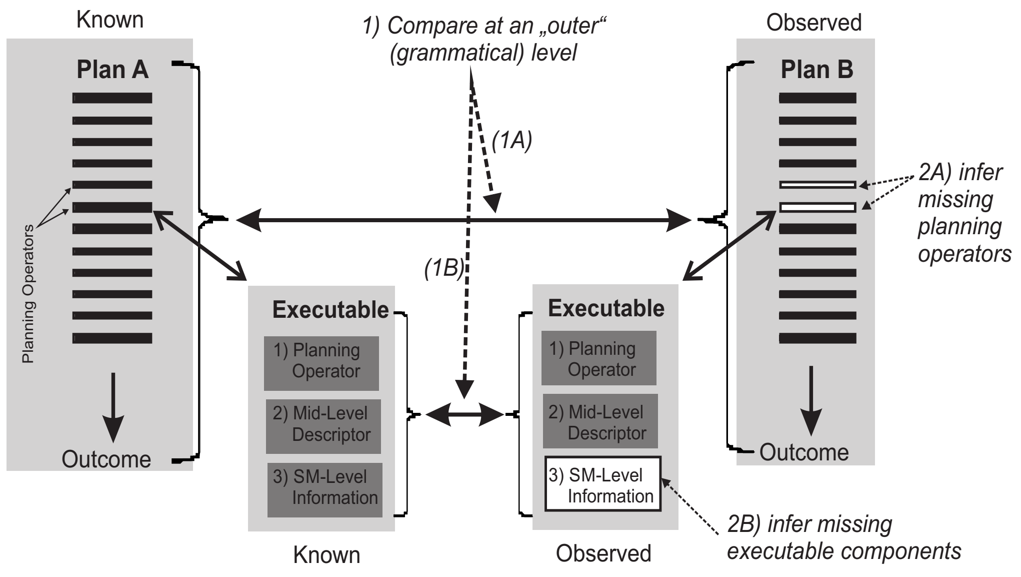


Figure 2: Schematic of structural bootstrapping.

Implementing Structural Bootstrapping at different levels

Figure 2 shows a schematic representation of the bootstrapping processes implemented here. A known plan A (left) consists of a set of planning operators (black bars) and each has attached to it an Executable consisting of the planning operator itself, a mid level descriptor and sensorimotor level information. The plan, being executed, also has a certain “outcome”, which can be considered as the goal of this action sequence. An observed plan B (right) of a similar action (with similar goal), will normally consist of many planning operators which are identical or highly similar to the ones of the known plan and also the outcome will be similar or the same. Still some planning operators may be dissimilar and hence unknown to the agent (white bars). In the same way, individual newly observed Executables (right) may contain unknown components (white). The goal of bootstrapping is to fill in all this missing information. To the end, first (1) the respective entities, Plans (1A) or Executables (1B), will be compared at an “outer”, grammatical level to find matching components. This way, in the second step one can try to infer the respective missing entities, planning operators (2A) or components of the Executables (2B).

Hence, a *central statement* is that structural bootstrapping always “propagates downward”. It uses type-similarities of entities from one level above to define the missing syntactical elements of the currently queried (lower) level. Plan similarities are used to infer planning operators, Executable similarities to infer Executable parameters such as objects, trajectories, forces, poses, and possibly more.

The main difficulty for implementing structural bootstrapping is to define appropriate scaffolds on which the bootstrapping can be based where – as described – the goal is to create novel information by generative processes which compare existing knowledge with newly observed one, without having to perform an in-depth analysis.

In the following we will now provide the details of the performed experiments, where we will show six different examples of structural bootstrapping for the different layers. These examples should allow the reader to more easily understand the so-far still rather abstract concept of structural bootstrapping.

Setup, procedures and specific problem formulation

Scenario (task)

ARMAR operates in a kitchen scenario. The task for the robot is to pour two ingredients (e.g. flour and water) and mix them together to obtain batter. For this the robot has the required knowledge to do it in one specific way (by using an electric mixer), but will fail whenever it should react flexibly to a changed situation (e.g. lack of the mixer). The goal of this work is to show that bootstrapping will quickly provide the required knowledge to successfully react to such a change. This process is based on observing a human providing an alternative solution (stirring with a spoon) where bootstrapping lead to the "understanding" of the meaning of objects and actions involved.

Prior knowledge

As bootstrapping relies on existing knowledge we have provided the robot with several (pre-programmed) Executables and we assume that the robot knows how to:

- pick up an object;
- put down an object;
- pour an ingredient;

- mix with an electric mixer.

In addition, robot has learned earlier to execute one apparently unrelated action, namely:

- wipe a surface with a sponge [25, 26, 27].

Furthermore the robot has a certain type of object memory where it has stored a set of objects together with their roles, called the *Repository of objects with attributes and roles (ROAR).* This prior knowledge can be inserted by hand or by prior experience. It allows objects to be retrieved by their attributes, and attributes of novel objects to be inferred, based on proximity in a low-dimensional, Euclidean space in which both, objects and attributes, reside [18].

The following entries exist in the ROAR:

- Sponge, rag, brush = objects-for-wiping with outcome: clean surface
- Mixer tool ends, whisks, sticks = objects for mixing with outcome: batter or dough.

Furthermore we have endowed the machine with a few recognition procedures:

- The robot can generate and analyze the semantic event chain (SEC) structures of observed (and own) actions by monitoring an action sequence using computer vision. Thus, the machine can recognize known actions at the SEC level [24, 17].
- The robot can recognize known objects (tools, ingredients, batter) using computer vision [28, 29, 30].
- The robot can explore unknown object haptically [31] and extract object features such as deformability and softness [32, 33, 25]

Problem definition

The problem(s) to be solved by structural bootstrapping are defined by several stages as spelt out next:
Normal System Operation: If all required entities are present (mixer, ingredients, containers, etc.) the robot can make a plan of how to make batter

and also execute it.

System Break-Down: Planning and execution will fail as soon as there is no mixer.

Alternative: The robot observes a human making batter by stirring the dough with a spoon.

Goal: The robot should find a way to understand the newly observed action and integrate it into its knowledge base and finally be able to also execute this.

Problem: The robot has no initial understanding of

- the planning requirements,
- the objects involved, and
- the movement patterns seen

in the newly observed stirring action. For example the robot does not know how to parameterize the rhythmic trajectory. Also, it does not know what a spoon is. Furthermore, the robot does not have any planning operator for stirring with a spoon in its plan-library.

Requirement (for the purpose of this study): The process of understanding the new action should happen without in-depth analysis of new actions constituents (hence without employing exploration based processes) but instead by using bootstrapping.

Methods - Short Summary

To not extend this paper unduly, methods are only described to the details necessary to understand the remainder of this paper. References to specific papers are provided where more details can be found.

Planning Methods

In this project, we are using the so-called Combinatory Categorial Grammars (CCGs) [16] to address the planning problem. CCGs are in the family of *lexicalized* grammars. As such they push all domain specific information into complex categories and have domain independent combinators that allow for the combination of the categories into larger and larger categories. As we have already alluded to, at the planning level, structural bootstrapping is a specialized form of learning new syntactic categories for known actions. A number of different methods have been suggested for this in the language learning literature [34, 35] for this project however we will be applying a variant of the work by Thomford [36]. However, we note that to do the kind of learning that we will propose it will be critical that the same syntactic knowledge, which is used by the system to plan for action, is also used to recognize the plans of other agents when observing their actions. This is not a new idea, however, there are very few AI planning and plan recognition systems that are able to use the exact same knowledge structures for both tasks.

Imagine that, as in our example, the high level reasoner knows about a plan to achieve a particular goal. It knows all of the actions that need to be executed, and for each action has encoded as CCG categories the knowledge necessary to direct its search for the plan. Further we suppose the same knowledge can be used to parse streams of observations of actions in order to recognize the plan being executed by others.

Now suppose the agent sees the execution of another plan that achieves the same goal. Let us assume that this new plan differs from the known plan in exactly one action. That is, all of the actions in the new plan are exactly the same as the actions in the known plan except for one action. Since the agent knows that the plan achieved the same goal, and it knows the CCG categories for each action that would be used to recognize the original plan, it is not unreasonable for the agent to assume that the new action should be assigned the same CCG category as its opposite action in the known plan.

If this addition is made to the grammar the agent now knows a new plan to achieve the goal and will immediately know both how to recognize others executing the plan and how to build the new plan for the goal itself (at the higher "abstract" level). The system will have performed structural bootstrapping at the planning level.

In this case, the system will have leveraged knowledge about the outcome of the observed plan being the same as the previously known plan, along with syntactic knowledge about how the previously known plan was constructed to provide new syntactic knowledge about how to construct and recognize the new plan.

Methods for the Mid-Level: Semantic Event Chains (SECs)

Semantic Event Chains [24, 17] encode in an abstract way the sequence of events that occur during a complex manipulation. They are used for two purposes: (1) Every event provides a specific temporal anchor point, which can be used to guide and temporally constrain the above described scene and motion analysis steps. And (2) the SEC-table itself (see Fig. 3 b), is used to define the mid-level of an Executable.

Fig. 3 shows the corresponding event chains extracted for a *stirring* action. SECs basically make use of image sequences (see Fig. 3 a, top) converted into uniquely trackable segments. The SEC framework first interprets the scene as undirected and unweighted graphs, nodes and edges of which represent image segments and their spatial touching or not-touching relations, respectively (see Fig. 3 a, bottom). Graphs hence become semantic representation of the relations of the segments (i.e. objects, including hand) presented in the scene in the space-time domain. The framework then discretizes the entire graph sequence by extracting only the main graphs, which are those

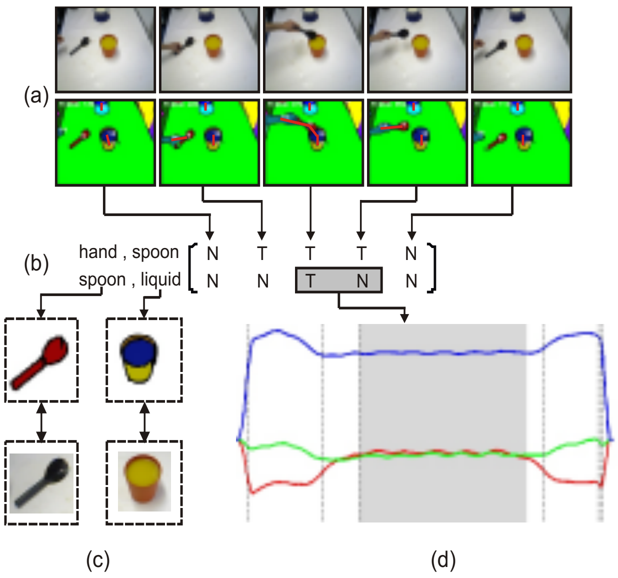


Figure 3: A real action scenario: *"Stirring liquid with a spoon"*. (a) Sample original key frames with respective segments and graphs. (b) Corresponding SEC where each key frame corresponds to one column. Possible spatial relations are N, T, and A standing for *"Not-touching"*, *"Touching"*, and *"Absence"*, respectively (*A* does not happen here.). Shaded box shows a sample relational transition. (c) Object identities derived from segments (d) Complete trajectory information for the hand. Trajectory segment for the time-chunk covered by shaded box in (b) is indicated in gray color.

where a relation has changed (e.g. from not-touching to touching). Each main graph, thus, represents an essential primitive of the manipulation. All extracted main graphs form the core skeleton of the SEC which is a sequence table (the SEC-table), where columns correspond to main graphs and rows to the spatial relations between each object pair in the scene (see Fig. 3 b). SECs consequently extract only the naked spatiotemporal relation-patterns and their sequentiality, which then provides us with the essence of an action, because SECs are invariant to the followed trajectory, manipulation speed, or relative object poses.

Columns of a SEC represent transitions between touching relations. Hence, they correspond to decisive temporal moments of the action and, consequentially, they allow now to specifically pay attention "at the right moment when something happens" to additional action relevant information (such as objects, poses, and trajectories). Fig. 3 (c-d)) illustrate syntactic elements of the manipulation. Manipulated objects, e.g. spoon and liquid, are extracted from the rows of event chains, i.e. from the nodes of the main graphs. Temporal anchor points provided by SECs can also be used to segment the measured hand-trajectory into parts for further analysis.

Sensorimotor Methods

Sensory Aspects: Visual scenes are analysed to recognize objects and their attributes, measure movement trajectories, and record object poses.

Basic object and pose recognition is performed in a straight-forward way using pre-defined classes of the different objects which occur during the actions of "stir", "wipe", and "mix" and in addition adding some distractor objects (e.g., cups, knifes, etc.). Any suitable method can be used for object detection, recognition, and pose estimation; such as edge-based, statistical shape representations [28, 29, 30, 37].

Another important aspect is object recognition for the construction of the repository of objects with attributes and roles (ROAR).

Our primary input for the ROAR consists of a table such as the one shown in Table 1.

Objects and attributes are (discrete) labels; values can be categorical, discrete or continuous. Examples of objects are "bowl" or "knife"; examples of attributes are "cuts", "food", "is elongated", "gripper orientation for grasping", "fillable", etc. We then use Homogeneity Analysis to project ob-

	Attribute 1	Attribute 1	Attribute 1
Object A	$Value_{A,1}$	$Value_{A,2}$	$Value_{A,3}$
Object B	$Value_{B,1}$	$Value_{B,2}$	$Value_{B,1}$

Table 1: ROAR encoding

jects and (attribute) values into the same, low-dimensional, Euclidean space (the *ROAR space*) [18]. This projection is set up such that:

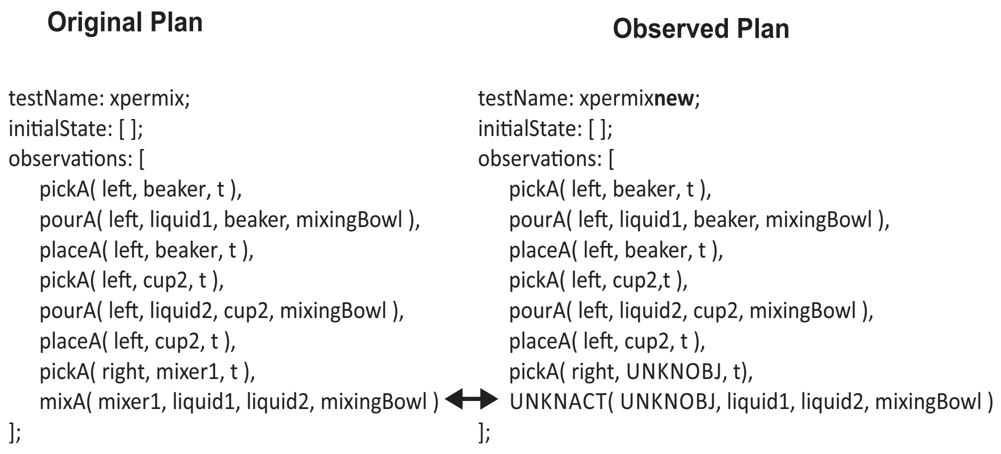
- Objects that exhibit similar attribute Values are located close together,
- Objects that exhibit dissimilar attribute Values are located far apart,
- Objects-as-such are close to their attribute Values.

Euclidean neighborhood relations allow us to make the following general types of inference:

- Attribute value prediction: Say, we have an object of which we know some but not all attribute Values. We can predict missing attribute Values by projecting the object into the ROAR and examining nearby attribute Values.
- Object selection: Say, we have a set of required attribute values. We can find suitable objects in the vicinity of these Values in the ROAR.

Note we cannot generally expect that very complex object/attribute relations will be faithfully represented in a low-dimensional Euclidean space. While we are currently working on more powerful representations for such relations, this is a complex research issue [18, 38, 39, 40, 41]. For us the ROAR is at the moment just a viable way forward, which allows us to demonstrate different aspects of structural bootstrapping.

Motor Aspects: Trajectory information is encoded by Dynamic Movement Primitives (DMPs), which were proposed as an efficient way to model goal-directed robot movements [19]. They can be applied to specify both point-to-point (discrete) and rhythmic (periodic) movements. A DMP consists of two parts: a linear second order attractor system that ensures convergence to a unique attractor point and a nonlinear forcing term. The forcing term


Original Plan

```
testName: xpermix;
initialState: [ ];
observations: [
    pickA( left, beaker, t ),
    pourA( left, liquid1, beaker, mixingBowl ),
    placeA( left, beaker, t ),
    pickA( left, cup2, t ),
    pourA( left, liquid2, cup2, mixingBowl ),
    placeA( left, cup2, t ),
    pickA( right, mixer1, t ),
    mixA( mixer1, liquid1, liquid2, mixingBowl )
];
```

Observed Plan

```
testName: xpermixnew;
initialState: [ ];
observations: [
    pickA( left, beaker, t ),
    pourA( left, liquid1, beaker, mixingBowl ),
    placeA( left, beaker, t ),
    pickA( left, cup2,t ),
    pourA( left, liquid2, cup2, mixingBowl ),
    placeA( left, cup2, t ),
    pickA( right, UNKNOBJ, t),
    UNKNACT( UNKNOBJ, liquid1, liquid2, mixingBowl )
];
```

Figure 4: Comparing known with observed plan. The arrow indicates where there is a novel, unknown planning operator found in the new plan. This is also associated with an, as yet, unknown object (the spoon).

is normally given as a linear combination of basis functions that are defined along the phase of the movement. The basis functions are either periodic or nonzero only on a finite phase interval. The type of basis functions decides whether the DMP defines a discrete or a periodic movement. DMPs have many favorable properties, e. g. they contain open parameters that can be used for learning without affecting the overall stability of the system, they can control timing without requiring an explicit time representation, they are robust against perturbations and they can be modulated to adapt to external sensory feedback [19, 42].

Concrete Examples of Structural Bootstrapping

Structural Bootstrapping at the Planning Level

The existing plan of making batter with a mixer is compared to the observed sequence of actions during making batter with a spoon. Due to the fact that all sub-actions, but one, are identical between known-action and new-action the agent can infer that the unknown sub- action (stirring with a spoon) is of the same type as its equivalent known sub-action (mixing with a mixer). Hence the grammatical comparison of known with unknown action renders a new (syntactic) planning operator entry for the unknown sub-action. This process is very similar to syntactic bootstrapping as observed in child language acquisition. A semantic element enters here due to the same outcome of both actions being recognized as batter. We use CCG as our planning language and we employ the PKS planner [15] for the actual planning processes of ARMAR III.

The actual inference process makes use of the similarity of known plan with newly observed plan, where in our example all but one action are identical.

Figure 4 shows the comparison between a known (and executable) plan on the left and an observed new one (right). Structural (grammatical) one-by-one comparison shows that there is just one unknown planning operator present. When the plan recognizer is run on the observed plan it would result in the following explanation of those observations with the highest probability:

```
[ addIngC(left, liquid1, beaker, mixingbowl),
  addIngC(left, liquid2, cup, mixingbowl),
  pickC(left, UNKNOBJ, table),
  UNKNACT(left, UNKNOBJ, liquid1, liquid2, mixingbowl)]
```

Note, the category name for the previously unseen action is simply denoted as UNKNACT. This is a special purpose category used to complete the explanation when we have an action that has never been seen before.

Now the agent has been told (or can observe) that the observed plan is a plan that achieves makeBatterC (making batter), and we will assume that all of the actions in the observed plan are relevant to the plan. The agent's job is to infer a category to replace UNKNACT that allows the completing of the parse. If the agent wants to build a category to assign to the unknown action that will result in a complete plan with the goal of makeBatterC, all it needs to do is walk the explanation from right to left collecting the categories and adding them to the complex category in order. This will result in the unknown action being given the following category:

```
action: UNKNACT(hand, UNKNOBJ, ingredient, ingredient, bowl)
   [ ((( makeBatterC( 2, 3, 4 ))\
            {addIngC( 0, 2, obj(1), 4)})\
               {addIngC( 0, 3, obj(2), 4)})\
                  {pickC( 0, 1, table(1)) } ];
```

Note the agent also infers the types and co-reference constraints for the basic category's arguments from the plan instance. In the above definitions we have denoted those arguments to the basic categories by numbers indicating when an argument is bound to the same argument as the action. (i.e. All references to "0" in the category refer to the hand used in the action because it is the zeroeth argument for the action. Likewise all reference to "4" in the category refer to the bowl argument of the action since it is the fourth argument.)

This category would represent the most restrictive hypothesis about the plan structure since it will require both that the actions be executed in the same order (and we know the ingredients can be added to the plan in either order) and that all of the arguments that co-refer in the example plan must co-refer in future instances. In this case, it would require that the same hand be used for all of the ingredient adding and mixing which we know to be overly restrictive.

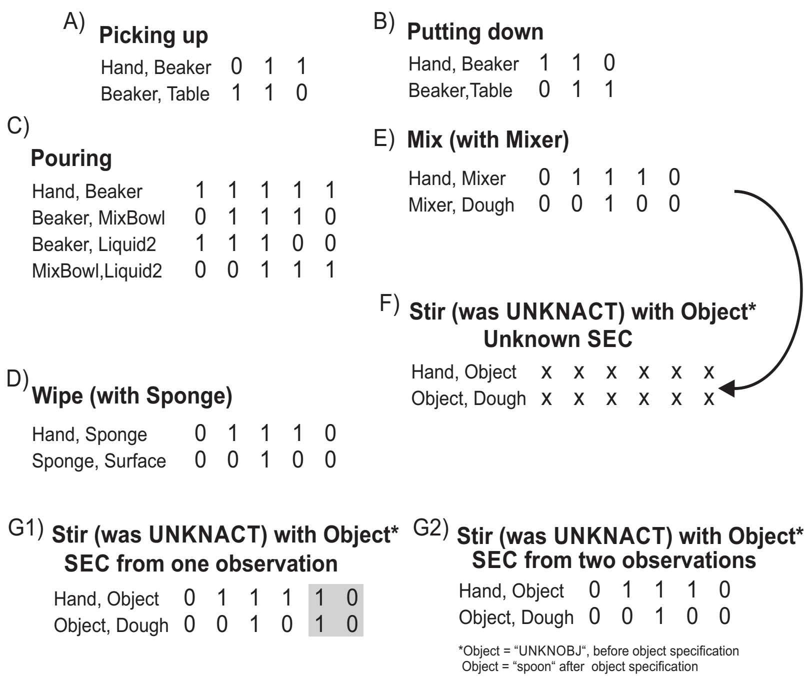
If we compare the new category to the category for the known mix action (mixA), we can see that the only differences are exactly in these overly restrictive areas:

1. The ordering of the categories for the add ingredient steps. The known category is more general allowing the ingredients to be added in any order while the new learned category has a required order.
2. The co-reference constraints are less restrictive in the known category. (Note the numbers indicating, which hand is to be used in the addIngC, are not the same so the plan would not enforce that the same hand be used.)

At this point, on the basis of the structural information provided by the parse and the action grammar, the agent has inferred that "UNKNACT" is equal to (or at least very similar to) "mixA" and the information can be entered directly into the planning grammar of the agent and forms the top-level of the corresponding new executable. We will, for convenience, from now on name it: "stir", hence we set:

`UNKNACT:=stir.`

While we have now added a new action to the planning grammar, still there is massive information lacking for designing the complete (new) executable for "stir", for example there is as yet no understanding existing about the UNKNOBJ (the spoon) and nothing is known about several other mid- and low-level descriptors.



A) **Picking up**

Hand, Beaker	0	1	1
Beaker, Table	1	1	0

B) **Putting down**

Hand, Beaker	1	1	0
Beaker,Table	0	1	1

C) **Pouring**

Hand, Beaker	1	1	1	1	1
Beaker, MixBowl	0	1	1	1	0
Beaker, Liquid2	1	1	1	0	0
MixBowl,Liquid2	0	0	1	1	1

D) **Wipe (with Sponge)**

Hand, Sponge	0	1	1	1	0
Sponge, Surface	0	0	1	0	0

E) **Mix (with Mixer)**

Hand, Mixer	0	1	1	1	0
Mixer, Dough	0	0	1	0	0

F) **Stir (was UNKNACT) with Object***
Unknown SEC

Hand, Object	x	x	x	x	x	x
Object, Dough	x	x	x	x	x	x

G1) **Stir (was UNKNACT) with Object***
SEC from one observation

Hand, Object	0	1	1	1	1	0
Object, Dough	0	0	1	0	1	0

G2) **Stir (was UNKNACT) with Object***
SEC from two observations

Hand, Object	0	1	1	1	0
Object, Dough	0	0	1	0	0

*Object = "UNKNOBJ", before object specification
Object = "spoon" after object specification

Figure 5: Several important SECs, which occur during the different actions. Headlines (bold lettering, like "Picking up", etc.) denote the type-specifiers of the different SECs. Note, sometimes objects can change. E.g. "Beaker" can be replaced by "Cup2". **A-E)** error-free archetypical SECs from known actions. **F)** So-far unspecified SEC. **G)** SECs from the unknown action extracted from observation of the human performing it. Hence these SECs might contain errors. **G1)** one observed case, **G2)** two observed cases. (In human terms: G1 corresponds to a case where the spoon had intermittently been pulled out from the dough (grey box), whereas for G2 it always remained inside until the action terminated.)

Structural Bootstrapping at the Mid-Level

At the mid-level, we need to define the correct SEC for "stir". Figure 5 shows SECs for several actions where (F) represents the so-far unknown SEC for "stir". Please, ignore panels (G) for a moment. Note, to be able to treat these tables numerically the intuitive notations from Figure 3 for non-touching "N" and touching "T" are now changed to "0" and "1" in Figure 5.

Structural bootstrapping at the mid-level uses as the "outer", grammatical scaffold the type-similarity of the planning operators (here "stir" and "mix") ascertained above. Hence we know that UNKNACT=stir.

Following this conjecture the agent can now with a certain probability assume that so-far unknown SEC for “stir” ought to be identical (or very similar) to the known one from “mix” and use the “mix”-SEC to define the mid-level (the SEC) for the Executable of “stir”. The arrow indicates that the SEC from panel (E) should just be transferred to fill the unknown SEC in (F) with the same entries.

There is a second line of evidence which supports this. Panels (G1) and (G2) represent the actually observed SECs of the stirring action here from a total of three observations of a human performing this. The SEC in panel (G1) had been observed once and the other twice. By comparing these SECs, the robot can with some certainty infer that the transfer of (E) to (F) was correct, because the more often observed SEC in (G2) corresponds to it, while the SEC from panel (G1) might be noisy as it is a bit different. As shown in an earlier study [24, 17], more frequent observations are likely to confirm this even more, but were not performed with the current setup.

Structural Bootstrapping at the Sensorimotor Level

Bootstrapping at the this level is used by the agent to find out how stirring is actually done (motion patterns), what the meaning of “UNKNOBJ” is, and which other objects might have a similar meaning. Before going into details we can quickly state that at the sensorimotor level several bootstrapping processes can be triggered. We note that bootstrapping is a probabilistic process and things can go wrong, too. One such example is, hence, also included. We find that the following processes are possible:

1. **Motion**
   (a) Bootstrapping from SEC-similarities [“wipe” and “stir”] to define the motion patterns for “stir”.
2. **Objects**
   (a) Bootstrapping from SEC-similarities [“wipe” and “stir”] into the new action. Here arriving at a false conjecture that “sponges” could be used for mixing.
   (b) Bootstrapping from SEC-similarities [“mix” and “stir”] from the repository of objects with attributes and roles (ROAR) into the new action seeking different objects that could potentially be used for mixing.
   (c) Bootstrapping from SEC-similarities [“mix” and “stir”] from the new action into the ROAR, entering the “spoon” into the category of objects for mixing.

To address the sensorimotor level the agent has to bootstrap from the mid-level downwards. It can do this by comparing the *type-similarities* of the different SECs. For this essentially one calculates a sub-string comparison of the rows and columns between one SEC and any other [24, 17]. We obtain that "stir" and "mix" as well as "stir" and "wipe" are 100% type-similar (compare panels D, E , and G2 in Figure 5), whereas "stir" and "pour" are only 52% similar, etc. Thus, the agent can infer that syntactical elements from "mix" and "wipe" might be used to define missing entities at the sensorimotor level of the Executable.

1a) Motion: Bootstrapping from SEC-similarities "wipe" and "stir" into the new action for completing motor information

Here we make use of the fact that the SEC for stir is very similar to the known one from wipe. Figure 6 shows the SECs and the different trajectories recorded from human observation for both actions. Note that for "wipe" the complete motor encoding is known and provided by the respective DMP parameters.

We have in our data-base the following description for "wipe": Since wiping is essentially a rhythmic movement, we use periodic dynamic movement primitives to specify the required behavior [27]. Periodic DMPs are defined by the following equation system [19]

$$\begin{aligned} \dot{z} &= \Omega\alpha_z(\beta_z(g-y)-z)+f(\phi), && (1)\\ \dot{y} &= \Omega z, && (2) \end{aligned}$$

In the above equations, g is the anchor point of the periodic movement. The nonlinear forcing term f is defined as

$$\begin{aligned} f(\phi,r) &= \frac{\sum_{i=1}^{N} w_i\Psi_i(\phi)}{\sum_{i=1}^{N}\Psi_i(\phi)}r, && (3)\\ \Psi_i(\phi) &= \exp\left(h_i\cos(\phi-c_i)-1\right), \end{aligned}$$

where the phase ϕ is given by

$$\dot{\phi} = \Omega. \quad (4)$$

Here we assume that a complete parameterization of the DMP for wiping has been learnt from earlier experiences. Given this the DMP can be easily modulated by changing:

- the anchor point g, which translates the movement,

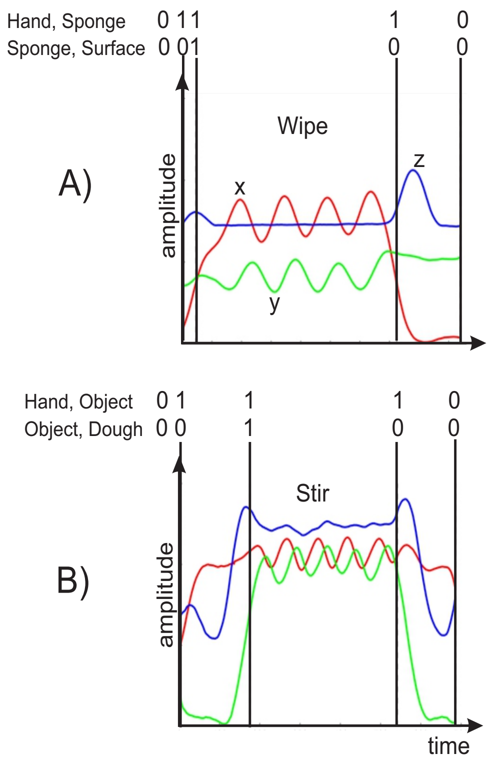


Figure 6: Bootstrapping motor information. SECs (top) and trajectories (bottom) for x, y, and z coordinates in task space are shown for **(A)** wipe and **(B)** stir.

- the amplitude of oscillation r,
- the frequency of oscillation Ω.

These variables can be used to immediately adapt the movement to sensory feedback.

Bootstrapping progresses by using the concept of temporal anchor points, which are those moments in time when a touching relation changes (from 0 to 1, or vice versa). These anchor points divide the trajectories in a natural way (shown by the vertical lines in the figure.)

Bootstrapping now *just copies* the complete DMP information from "wipe" to the Executable of "stir" between the respective anchor points only leaving the constraint-parameters (e.g. amplitude) open as those are given by the situation (mainly the size of the bowl wherein to stir). Thus, the agent assumes that it can use the motor encoding from "wipe" in an unaltered way to also perform “stir”. We know from own experience that this largely holds true. Here we can also clearly see the advantages of bootstrapping: we do not need any process that *extracts and generalizes* motor information from the observed example(s) of “stir” (a process which could be more tediously performed by methods from imitation learning [43, 44, 45]). Instead we just copy. Clearly, the agent - like any young child - will have to ground this by trying out the stirring action (see the Discussion section for the “grounding-issues”). It will possibly then have to adjust the force profile, which is likely to be much different for wipe and stir. Still, all this is faster than learning the required motor pattern in any other way. The benchmark experiments below show this clearly.

2a) Objects: Bootstrapping from SEC-similarities “wipe” and “stir” into the new action for object use

The SEC-similarities between “wipe” and “stir” allow the agent to also (wrongly!) infer that the object for wiping (sponge) should be suitable for stirring, too. Note this may seem unexpected but can happen during any bootstrapping process due to its probabilistic nature. The use of just one single scaffold (here the SECs) is not strong enough to allow rigorously excluding such false conjectures. For this the agent needs to integrate additional information and, due to the fact that there is a repository of objects with attributes and roles (ROAR), it can indeed obtain evidence that there has been an error.

The agent knows that “stir” and “mix” are at the mid-level (SEC) type-similar action. It finds, however, that sponges are clearly outside the cluster of objects for mixing (Figure 7 A). This lowers the probability substantially that sponges should be used for mixing/stirring actions.

Interestingly, children will many times indeed over-generalize and use “unsuitable” objects for an intended action [46]. It is unknown how the brain represents this, but – clearly – their representation does apparently not yet contain the fine grained-ness of an adult representation.

2b) Bootstrapping from SEC-similarities “mix” and “stir” from the ROAR to find other suitable objects

Here the agent falls back (again) on the similarity of the new SECs of “stir” with the known one of “mix”. Due to this similarity, the agent knows that appropriate objects for the novel action might be found in the cluster of “objects for mixing” in the repository of objects with attributes and roles.

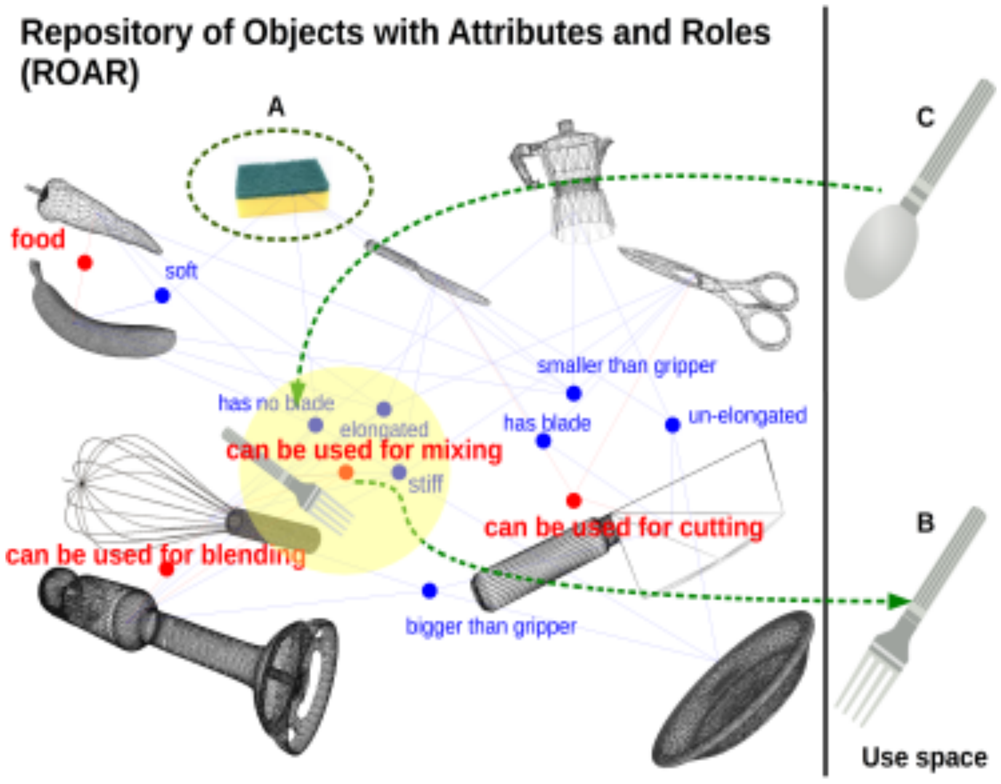


Figure 7: Bootstrapping object information. Graphical rendering of the repository of objects with attributes and roles (ROAR). Depicted are the metric distances between the different objects and the attribute values that describe their respective roles. **A)** The sponge is located far from the attribute value "can be used for mixing". **B)** Bootstrapping allows inferring that a fork, found close to the "mixing" attribute value, could be used also for "stir", as "mix" and "stir" are at the SEC-level type-similar. **C)** Following this SEC-similarity, a novel object (spoon) with unknown "mixing" attribute may be hypothesized useful for mixing by the ROAR and also due to other, known attribute values (such as shape, stiffness, and SEC characteristics of known, observed actions).

Hence it can ask the repository for a tool suitable for mixing and maybe locate it somewhere else on the table. Clearly this process will lead to an action relevant result only in those cases where the agent actually find such an object within reach. Then it can try to use this object for stirring, too. Again we can draw similarities to our own behavior. Generally this type of tool-replacement is found for a wide variety of actions where we "define" the tool according to its planned use. Our own generalization properties may here go far beyond what the ROAR offers to our artificial robotic agent, which is evident from situations where we "abuse" objects for entirely different purposes.

2c) Bootstrapping from SEC-similarities "mix" and "stir" from the new action into the ROAR to create a new entry

In the last step, the agent can perform one more bootstrapping procedure to augment the repository of objects with attributes and roles. For this it analyzes the outcomes of the actions realizing that batter is obtained from "mixing" and also from the unknown action of "stirring".

Thus, the agent can enter the new observed tool (spoon) into the ROAR and can then – by virtue of its resulting position in the ROAR – infer other, unobserved attribute values (uses), which is a bootstrapping effect. This way the repository will be extended by a novel entry following a single-shot experience. This step, however, does require a parametrization of the new object according to the features used for the ROAR.

Robotic implementation and benchmark experiments

Note, the actual bootstrapping processes happen "inside the machine" and any demonstration will, thus, only show that "the robot can do it now". To go beyond such mere visual inspection, one needs to show quantitative results on performance gain by bootstrapping, which will be presented in the next sections, below.

Still, a complete robotic implementation of these processes is currently being performed using the our robot systems [47]. For brevity, we will here show one central part of this implementation demonstrating the required transfer of human action knowledge (Fig. 8 A) onto the robot. This is the initial step needed to set up action knowledge in the machine before any bootstrapping can happen. The robot acquires here the knowledge to perform mixing with a mixer.

To better be able to extract object relations we have here used a Vicon-based motion capture system from which we immediately get error-free Semantic Event Chains (Fig. 8 B). The complete action relevant information is extracted at the respective key frames and encoded into the required Executables (Fig. 8 C), which can be used by the robot to reproduce this action (Fig. 8 D). The complete experiment is described elsewhere [48].