Deliverable number:	D5.4
Deliverable Title:	Demonstrator of intermediate integration of the system
Type (Internal, Restricted, Public):	PU
Authors:	C. Schou, O. Madsen, J. Rytz, L. Kiforenko, L. Bodehagen, N. Krüger, D.Vitkute-Adzgauskiene, I Markievicz, A.Ude, A. Gams, A. Kramberger, B. Ridge, H. Langer, D. Nyga, G. Lisca, M.Tamosiunaite, F. Wörgötter , D. Chrysostomou
Contributing Partners:	All

Project acronym:	ACAT
Project Type:	STREP
Project Title:	Learning and Execution of Action Categories - Update
Contract Number:	600578
Starting Date:	01-03-2013
Ending Date:	30-04-2016

Contractual Date of Delivery to the EC:	02-03-2015
Actual Date of Delivery to the EC:	28-04-2015

Content

1. Executive summary

This document presents an overview of the demonstrators for CHEMLAB and IASSES at the end of month 27 of the project. Agreed with the Commission, submission of this deliverable comes together with the second year report (and not at month 24 – as originally planned), due to the fact that this better reflects the status of the demonstrators as planned for the second year review of ACAT.

Note, some text of the WP5 in the PPR of Year 2 is largely recompiled from this deliverable (which is explicitly indicated in the PPR). This is done for the sake of consistency between the two documents.

Our main demonstrator in year 2 is the one for the IASSES scenario which we show on a real robot and where inputs from five partners of the consortium have been integrated. The demonstrator for the CHEMLAB scenario will be shown as a video and is integrated through Action Data Table (ADT) usage. In an Appendix, we describe the different methods used for sub-symbolic ADT-compilation, required for the demonstrators.

2. Introduction

For year two review meeting we are preparing two demonstrators: IASSES scenario and CHEMLAB scenario. The "main" demonstrator for year 2 is the AAU system based on Little Helper robot (IASSES scenario), which will be shown by a life demo. In addition, we will show a demonstrator from CHEMLAB scenario as a video.

ACAT – as pointed out in the first progress report – does not integrate at the execution level, but at the level of Action Data Tables (ADTs: formal representation of robotic actions used in ACAT project; presented in great detail in PPR 2, reporting on WP2). We advocate this as a major strength of the ACAT approach, because ADTs allow bridging between different execution engines, using "Compilation Processes" (as shown in the Appendix of this Deliverable) to create a new ADT, which then "just" needs to be interpreted locally by the used execution engine.

ADTs as means of robot-to-robot information transfer provide, thus, the main unifying structures, which are embodiment-free. The actual embodiment and its constraints enter only at the level of the execution engine(s).

The main argument for this approach is the potential use of ADTs by the community that will have a far higher threshold if you demand that users need to change their complete robotic execution setup as compared to the need to write just an "ADT interpreter" which translates ADT information into their robotic execution framework as needed.

Hence not everyone has to use everything from the given ADTs. ADTs provide "information on demand", but do not force the user!

The strength of this attitude is currently demonstrated by the three partners (UGOE, AAU, and UoB) who are concerned with the actual robotic execution aspects and are making use of ADTs in this way. All of them use ADT information differently to feed their execution engines. Figure 1 below shows our integration approach. JSI plans to do this, too, in the near future; and SDU in simulation.

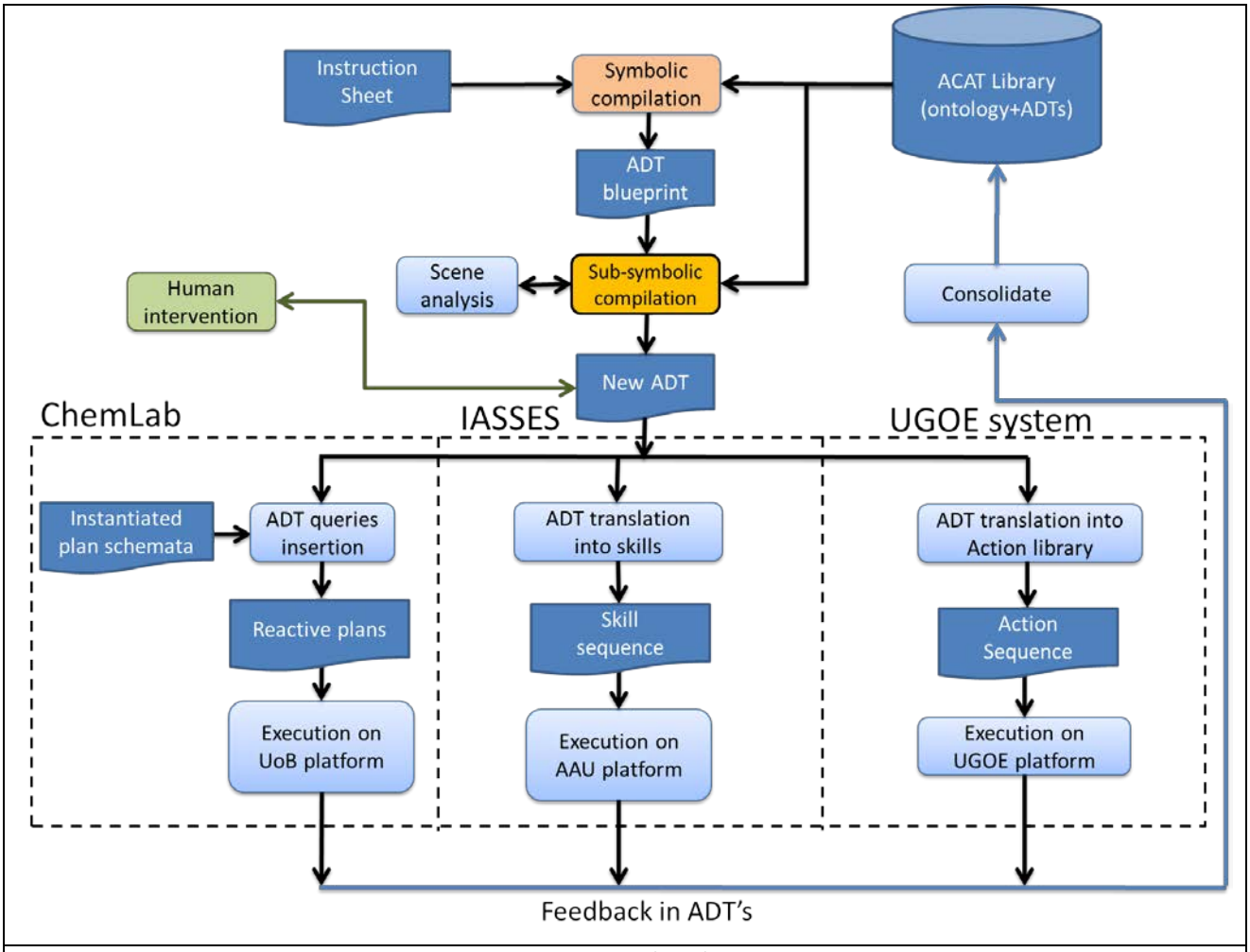

Figure 1. Integration based on ADTs

First the instruction sheet is symbolically analyzed (See also Periodic report year 2, WP3, section "Algorithms for textual completion of instruction sheets leading to an ADT blueprint") and names of action, main, primary, secondary object, etc. are found with the help of information held in the existing ADTs in the ACAT library. The obtained action and object names are inserted into an empty ADT by which we create the "ADT blueprint". The ACAT library is then queried a second time to perform filling-in of sub-symbolic information. Here scene context enters, for example to recalculate relative poses, etc. After this the "New ADT" exists. Ideally this is executable, but at this stage of the project sometimes additional information needs to be filled in by hand (Human intervention). The New ADT is then sent to the different Execution Engines. They are making use of (different) ADT information. The process completes by sending information back to the ACAT library, for example correcting the New ADT after execution if errors had been encountered. This way only checked New ADTs are stored in the ACAT library.

We note that in almost all cases, with very few exceptions, existing ADTs cannot be re-used as such. This will only happen if exactly the same situation is repeated as encoded by the ADT. Hence, the processes described in Figure 1 are generic and the situation that "New ADT" can be one-to-one equated to an existing ADT rarely happens.

Further we will describe in greater details two ACAT demonstrators based on the general integration scheme presented above.

3. Progress in the ACAT demonstrators

In the following we provide introduction of two ACAT demonstrators for IASSES and CHEMLAB scenarios.

3.1. IASSES Scenario

For IASSES scenario one conjoint demonstrators has been developed, where partners AAU, VDU, SDU, JSI and UGOE have made direct contributions to the different components (Figure 2).

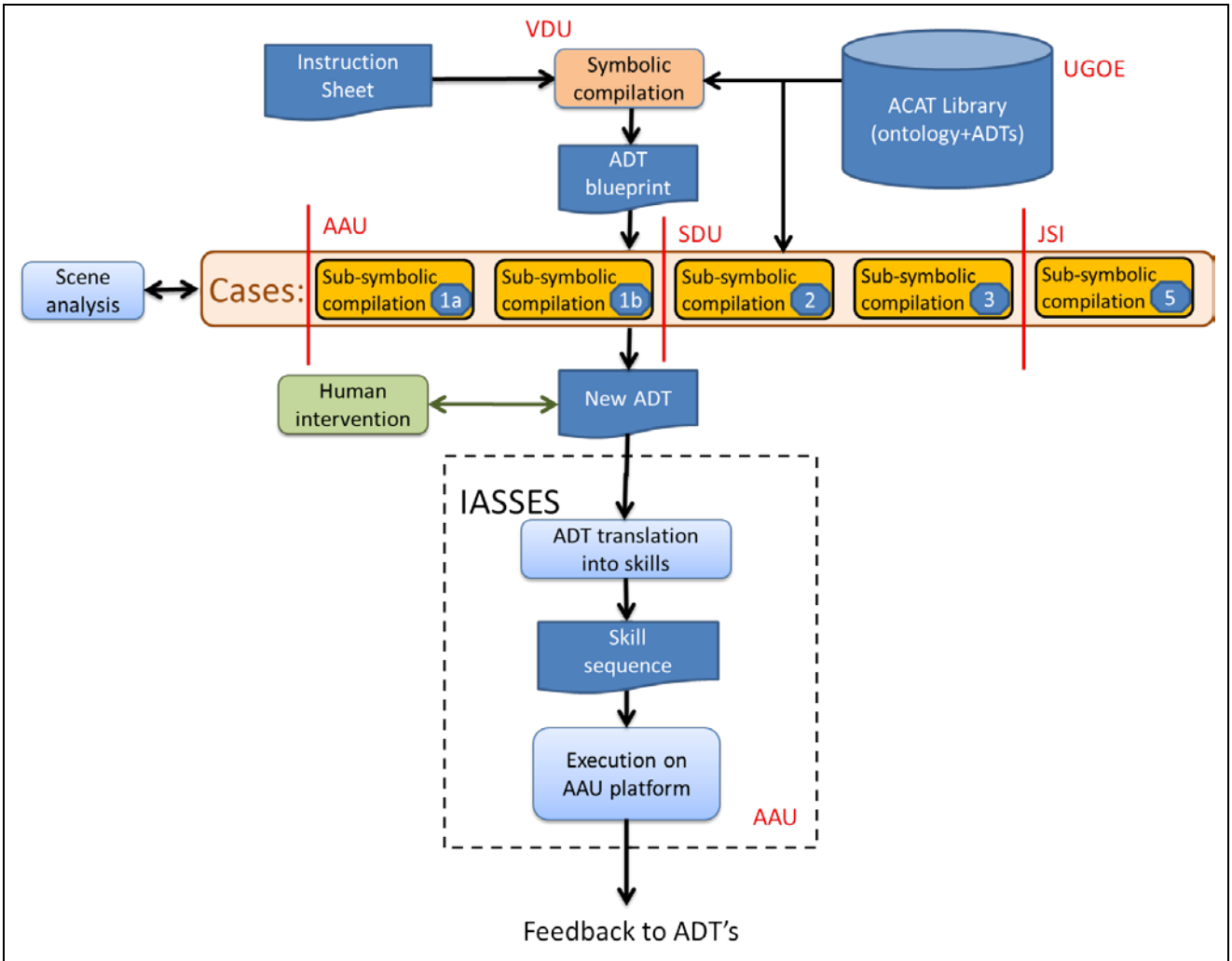

Figure 2. The variations of the IASSES demonstration. Sub-symbolic aspects are from WP3 (see there). Main partner contributions are indicated in red.

In this demonstrator we execute five different instructions in several variations ranging from straightforward execution of a previously filled ADT to combining ADT information and grasp planning, to sub-symbolic generalization based on several symbolically fully equivalent ADTs. The main aspect is to integrate from high-level symbolic compilation to low-level execution on the AAU's robot platform (Figure 2

shows how we re-use the general ACAT integration scheme from Figure 1). The five instructions involved in the demonstration are:

A) Take a rotor cap from conveyor and place it on fixture.
B) Take a rotor cap from conveyor and place it on a robot platform.
C) Take a rotor cap from the table and place it on a fixture.
D) Take a metal bottle from conveyor and place it on the robot platform.
E) Insert a shaft into a hole in a box.

The first two instructions as well as the last one have ADTs located in the ACAT database, however instructions (C) and (D) are "new" and do not have corresponding ADTs in the database. Different cases of sub-symbolic compilation: 1a, 1b, 2, 3, 5 indicated in Fig. 2 use instructions (A)-(E) in the following way:

- Cases 1a and 1b: we assume an ADT for instruction (A) exists in the A-cat library, and we get as a "new instruction" the instruction (A) for a second time; thus these are cases where we just re-execute an existing ADT. Case 1a is a complete re-execution (trivial case) and 1b is re-execution with changes in object poses.
- Case 2: We assume an ADT for instruction (A) exists in the A-cat library, but now we get a new instruction (C), we have not executed before.
- Case 3: We assume an ADT for instruction (B) exists in the A-cat library, but now we get a new instruction (D), we have not executed before.
- Case 5: We have multiple ADTs for instruction (E) in the A-cat library and we get as a "new instruction" the instruction (E) again but in a different situation. This is a case of sub-symbolic generalization.

The details of sub-symbolic compilation for the cases 1-5 are described in the Appendix, below.

In the following we will extend the discussion of case 5 (Instruction (E): "insert", in robotic terms "peg in hole") as this contains another interesting aspect, *statistical learning* where, with this demonstration we show that we can: 1. successfully generate an initial peg insertion operation from a number of examples stored in ADTs using statistical learning techniques (see Appendix), and 2. improve the speed of execution by successive performance of the peg insertion operation in the same configuration. The best execution after learning can be used to generate a new ADT describing the peg insertion operation for the new peg length.

3.2. CHEMLAB Scenario

For the CHEMLAB scenario the starting point of the demonstration is a natural language instruction text, which describes a chemical experiment. This natural language instruction is interpreted using the PRAC (Probabilistic Action Cores) system. The interpretation includes reasoning about the assignment of the most likely semantic role to noun phrases as well as inference on missing constituents, e.g., objects required for the execution of an action but not explicitly mentioned in the instruction text. The reasoning capabilities of the PRAC system are used to create a complete abstract (symbolic) parameter specification which then, in a next step, is transformed into an executable plan parameterization. This transformation step involves both manual and automated methods. The fully parameterized plan can finally be executed by the robot. The resulting logfiles serve as a database for deriving ADTs for the executed actions. Entire system architecture

is shown in Figure 3 below. ADT extraction from CHEMLAB execution logfiles serves as a connecting link to the other components of ACAT.

For the year 2 demonstrator a plan library has been developed comprising activities for conducting chemical experiments, such as prototypical plan schemata for actions “Placing”, “Adding”, “Pipetting”, “Opening”, “Closing”, “Operating a Centrifuge”. The plan execution has already been shown at a public demonstration in context of the “Ocean Sampling Day”. Log data of the experiments is in the process of being made accessible on the OpenEASE knowledge base and will serve benchmarking and evaluation. Selected prolog queries are being developed which are used to extract ADT entries from experience data stored in OpenEASE. Making a collection of ADTs accessible in OpenEASE is ongoing work.

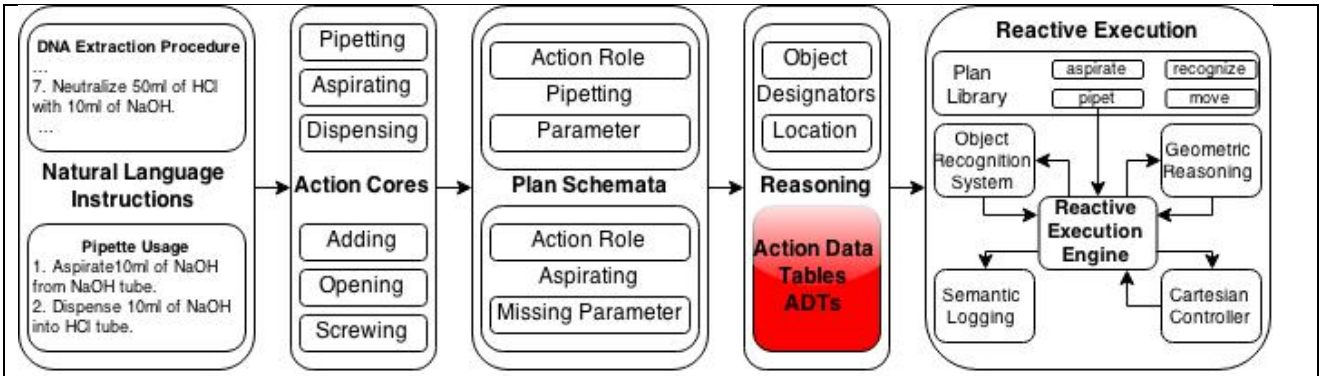

Figure 3, UoB execution architecture (copied from WP4)

4. APPENDIX: ADT filling with symbolic and sub-symbolic information

The following two sections are part of different deliverables but we summarize this here briefly to better allow understanding of the whole process depicted in Figure 1. Of direct relevance for the execution examples is section 4.3, which discuss the different cases.

4.1. Summary on symbolic compilation

Symbolic compilation aims at extracting symbolic names from instructions, which are: main action (i.e. the action that takes the central role in the instruction, e.g. it would be “place” in the instructions (A)-(D) above), main object (i.e. the object which is touched by the hand first, it would be “rotor cap” in instructions (A)-(C) and “metal bottle” in instruction (D)), primary object (i.e. the object which main object un-touched or touches first, e.g. it would be conveyor in instructions (A), (B) and (D) and “table” in instruction (C)) and secondary object (the object that is second touched or un-touched with the main object, “fixture” in instructions (A) and (C) and “robot platform” in instructions (B) and (D). In the simple examples as instructions (A) to (D) above the extraction is performed using dependency parsing of the instruction sentence. For more details see D3.1-update.

After the main action and main, primary and secondary objects are extracted, this information is placed on so called ADT blueprint: an otherwise empty ADT containing only the symbolic information just mentioned.

The ADT blueprint is then used as a search template for searching in the database of ADTs filled during previous robotic execution.

4.2. Summary on sub- symbolic compilation

Details of this are found in D3.2. In summary: After an ADT blueprint has been produced, by the process of **symbolic** compilation, we are searching for similar ADTs in the ACAT library and continue filling of the ADT blueprint with sub-symbolic information so as to obtain a finalized ADT. The ADTs are then translated into execution entities of different partner execution engines and executed on our robots.

Similarity of the ADT blueprint to the existing ADTs is calculated according to how well action name and object (main, primary, secondary) names match between ADT blueprint and existing ADTs. It is most straightforward to re-use existing ADT information when all the names match exactly (i.e., we need to execute the same action with the same objects we have already executed before). However, we attempt also partial information re-use in cases when a fully symbolically matching ADT cannot be found.

If the action as well main, primary, secondary object and tool names match completely between ADT blueprint and the best matching existing ADT, we can re-execute the existing ADT after having recalculated the poses of the manipulator in respect to new poses of objects in the scene. Note, we use vision methods to determine object poses in the scene. As all object coordinates are existing in the existing ADTs, we can obtain relative poses between any two entities indicated in the ADT. According to the touching and un-touching events recorded in the Semantic Event Chain (which is provided in the ADT as well), we can judge which objects are being approached by the manipulator (or by other objects) and re-calculate manipulator poses required in the new situation appropriately.

In addition to fully ADT-based information re-use, we also consider cases where additional sources of action information are used. E.g. in case grasp information is not considered reliable in the existing ADT, we combine the existing ADT information with a grasp planner based on 3D shape analysis. We also consider choosing ADTs with most similar sub-symbolic information and generalizing from those in case we have multiple symbolically fully matching ADTs.

In the next section we provide five different examples on sub-symbolic compilation which will be shown in year 2 Project demonstrators.

4.3. Examples of sub-symbolic compilation used for execution

Note, most of the text presented in this section is copied from WP3 report for reviewer's convenience.

We have designed a series of examples which show different cases of sub-symbolic compilation. All these examples will be shown as demonstrators at the year 2 review (some integrated into the IASSES system, the others as individual demonstrators on compilation).

Case 1 is the simplest compilation case where we directly re-execute an existing ADT when everything, included the poses of the objects, in the scene and in the existing ADT either fully match (1a, see Figure 4), or where we just have to perform a pose transform (1b, see Figure 5). This case is shown in the CHEMLAB

demonstrator for instructions of the type “pick and place” as well as in UGOE system for instructions of the type “pick and place” and “unscrew” (see also D3.2).

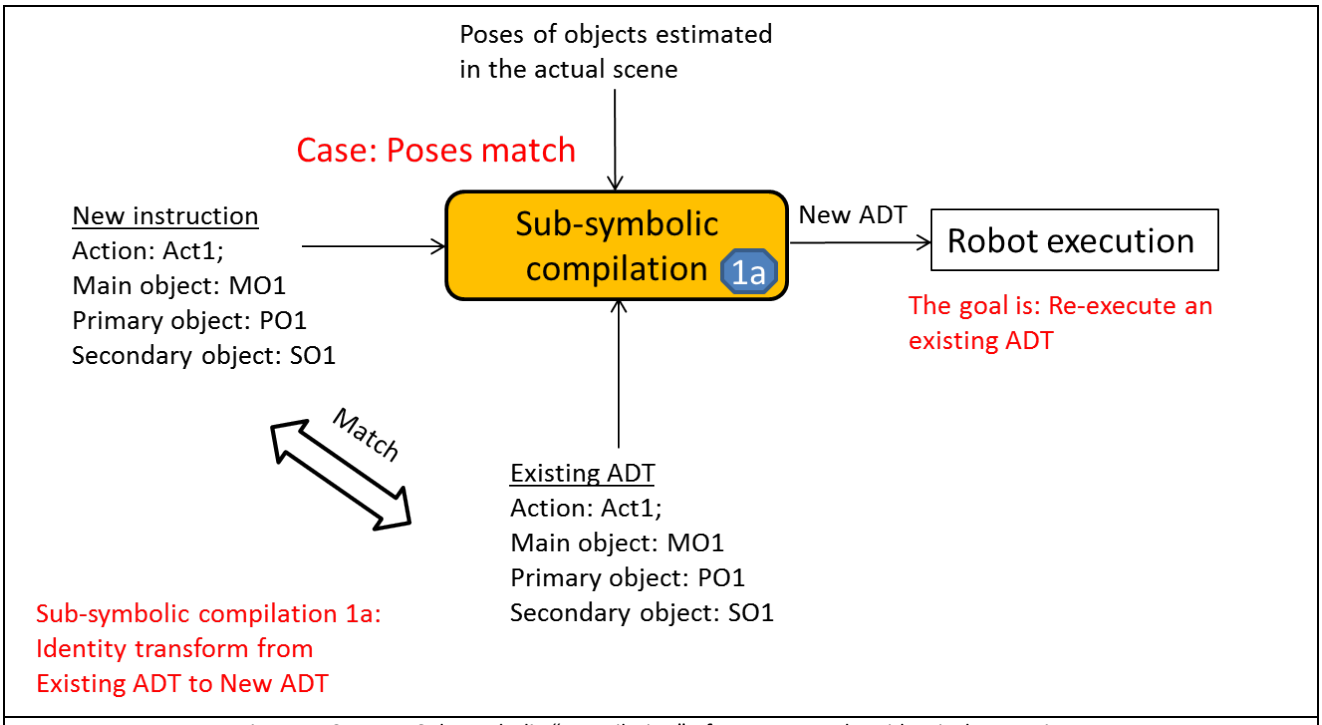

Figure 4. Case 1a: Sub-symbolic “compilation” of a new ADT when identical ADT exists.

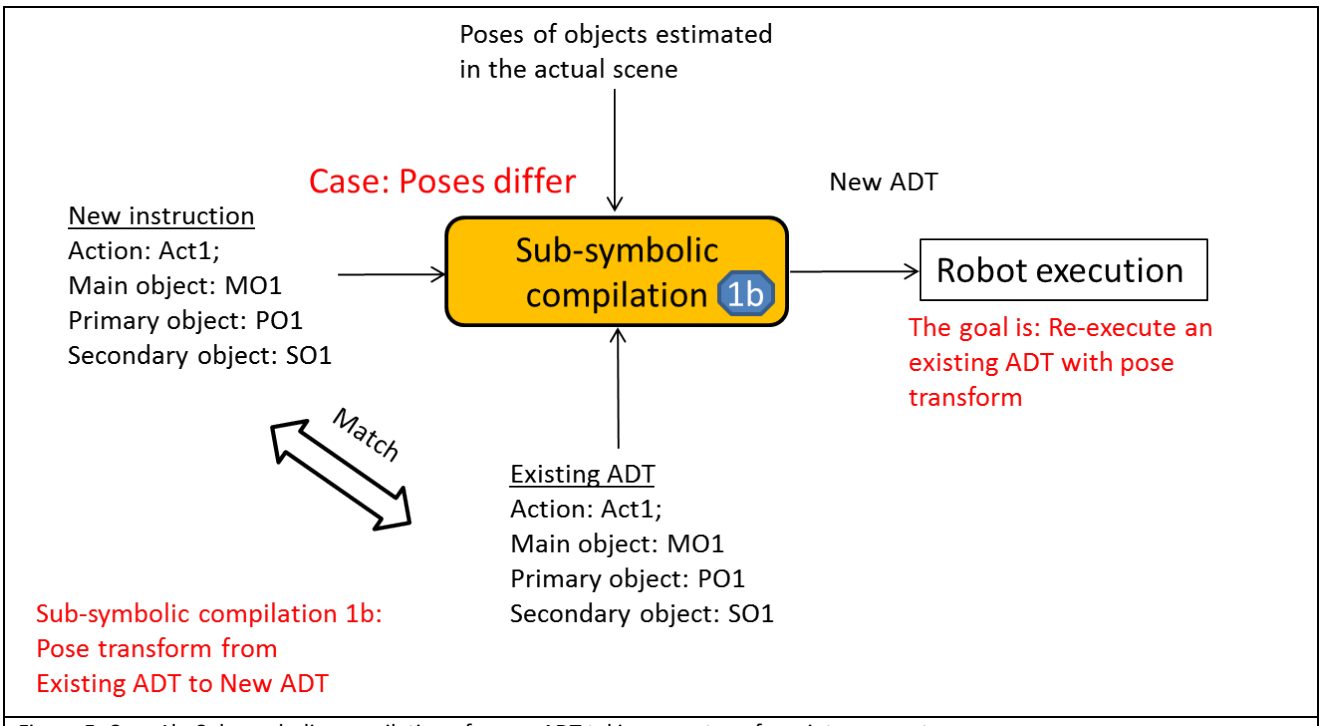

Figure 5. Case 1b: Sub-symbolic compilation of a new ADT taking pose transform into account.

In case 2 the symbolic names in the new instruction and the existing ADT match, except for the primary object. This makes the grasp information (at least in an industrial context) untrustworthy, so the existing

ADT information is re-used except for the grasp, which is calculated based on object 3D shape and scene context as measured by vision (see Figure 6). An example for that, which we also show in the IASSES demonstrator, is the following: we have an ADT for the instruction "Take a rotor cap from conveyor and place it on a fixture". The new instruction is "Take a rotor cap from table and put on a fixture". The primary objects "conveyor" and "table" between the existing ADT and the new instruction differ. Thus, we do not trust that grasping of the rotor cap is the same when having to take it from the conveyor as compared to taking it from the table. Therefore, we plan the grasp from scratch based on vision information, and re-use the "placing" part of the ADT. (Note, we could have risked re-using the original grasp, as the main object in both instructions is the same, but this can lead to troubles!).

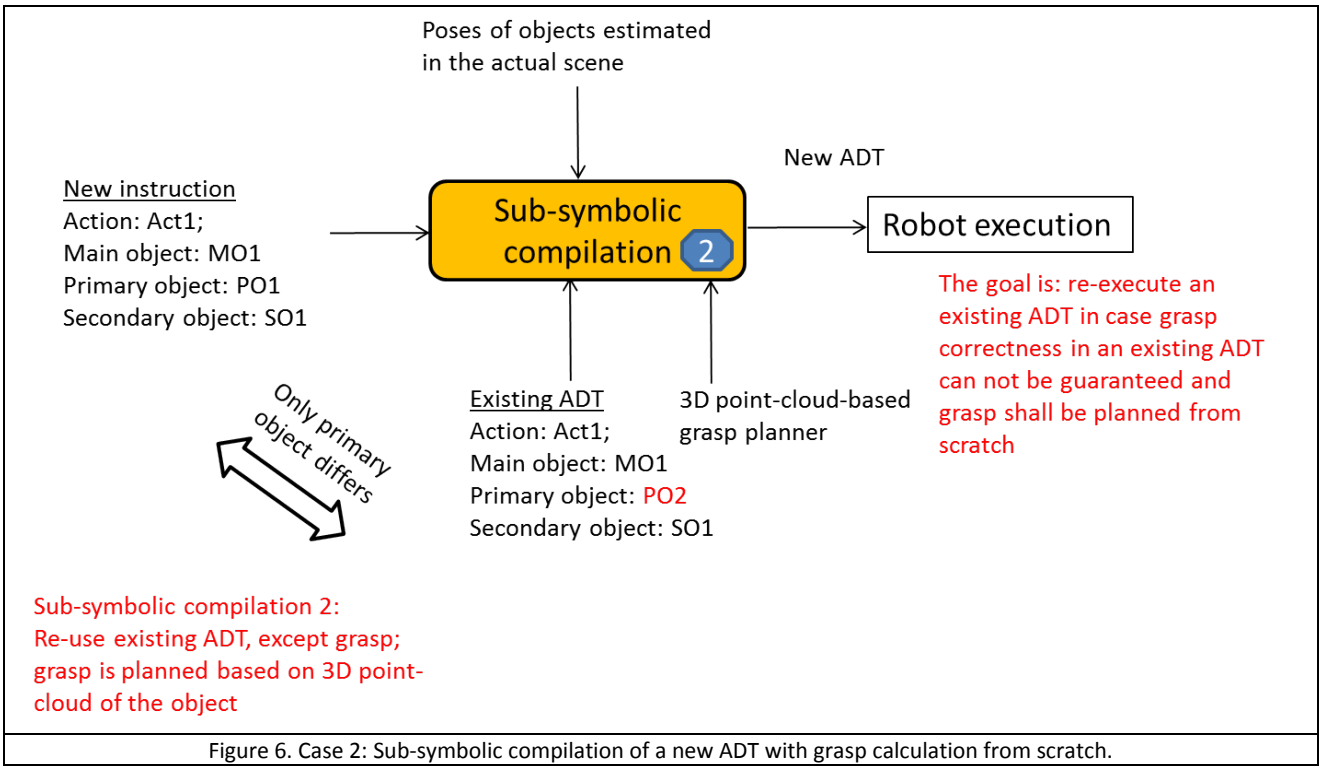

Figure 6. Case 2: Sub-symbolic compilation of a new ADT with grasp calculation from scratch.

In case 3 (Figure 7) the symbolic names in the existing ADT match to the ones in the instruction, except for the main object. Because a 3D model is always attached to the ADT for the main object, we check if the models of the new main object and the one attached to the existing ADT are similar. In case of similar models, we re-use the full ADT, as in case 1b. Here the example which we further will use in the IASSES scenario demonstrator is the following: we have an existing ADT for the instruction "Take a rotor cap from conveyor and put it on the robot platform". The new instruction reads "Take a metal bottle from conveyor and put it on the robot platform". Here the difference is only in the main object: "rotor cap" in the existing ADT and "metal bottle" in the new instruction. The check on 3D models (rotor cap in the existing ADT and metal bottle as provided together with the new instruction) shows that the models have high shape and size similarity, thus, we fully re-use information of the existing ADT.

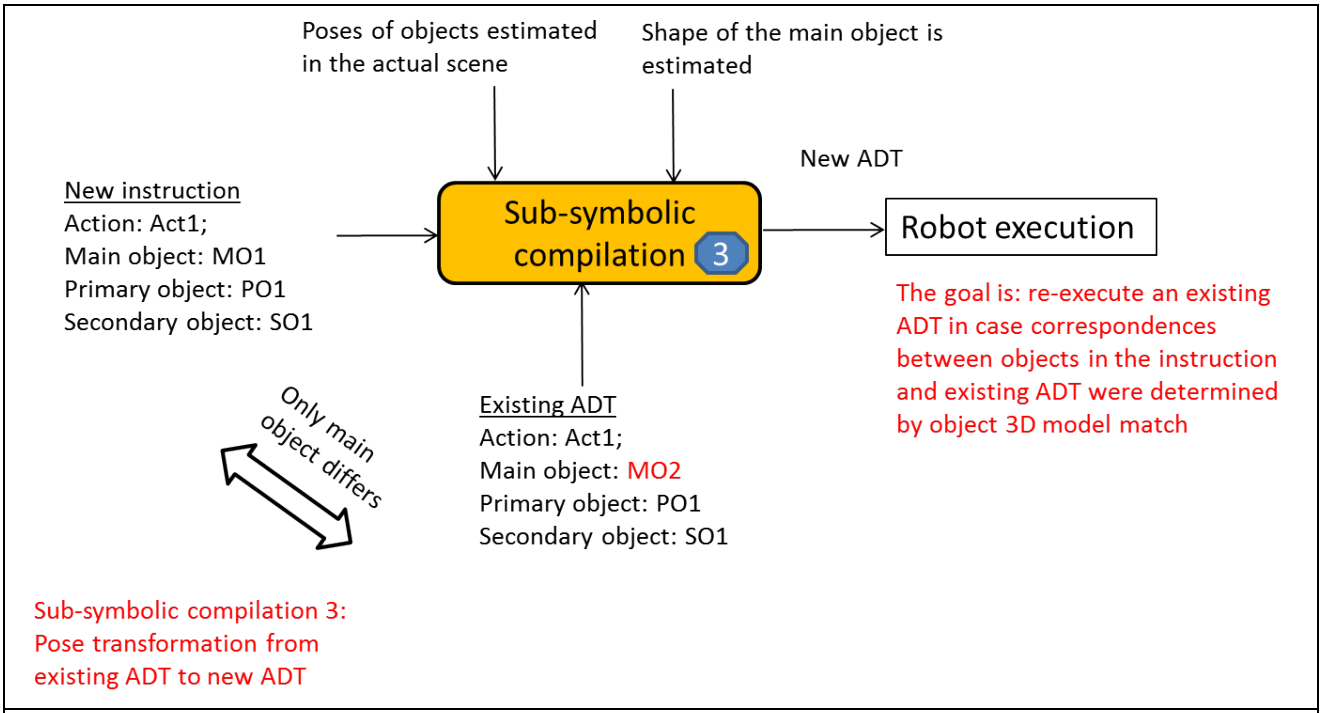

Figure 7. Case 3: Sub-symbolic compilation of a new ADT in case object models match.

Cases 4a and 4b represent a more complicated information reuse situation, where the entire information required for new instruction sub-symbolic compilation cannot be found on none of the existing ADTs, however can be combined from several existing ADTs. These cases will be shown as compilation demonstration performed on the UGOE system.

In the first case, the instruction to be executed is "Pick the measuring beaker and place it in the cup". We have an ADT for the same action PickAndPlace, but a different main object: “Picking a plastic bottle and placing it in a cup”. The missing information is how to approach and grasp the measuring beaker, which is different from the plastic bottle. We have another ADT, for another action: “Shake the measuring beaker”, in which the information how to approach and grasp the measuring beaker exists. Here the compilation means to take the first ADT, and replace the grasping information using the one from the “shaking ADT”. In this way we create an ADT for the desired instruction and use the resulting new ADT for execution. The schematic view for this type of compilation is presented in Figure 8.

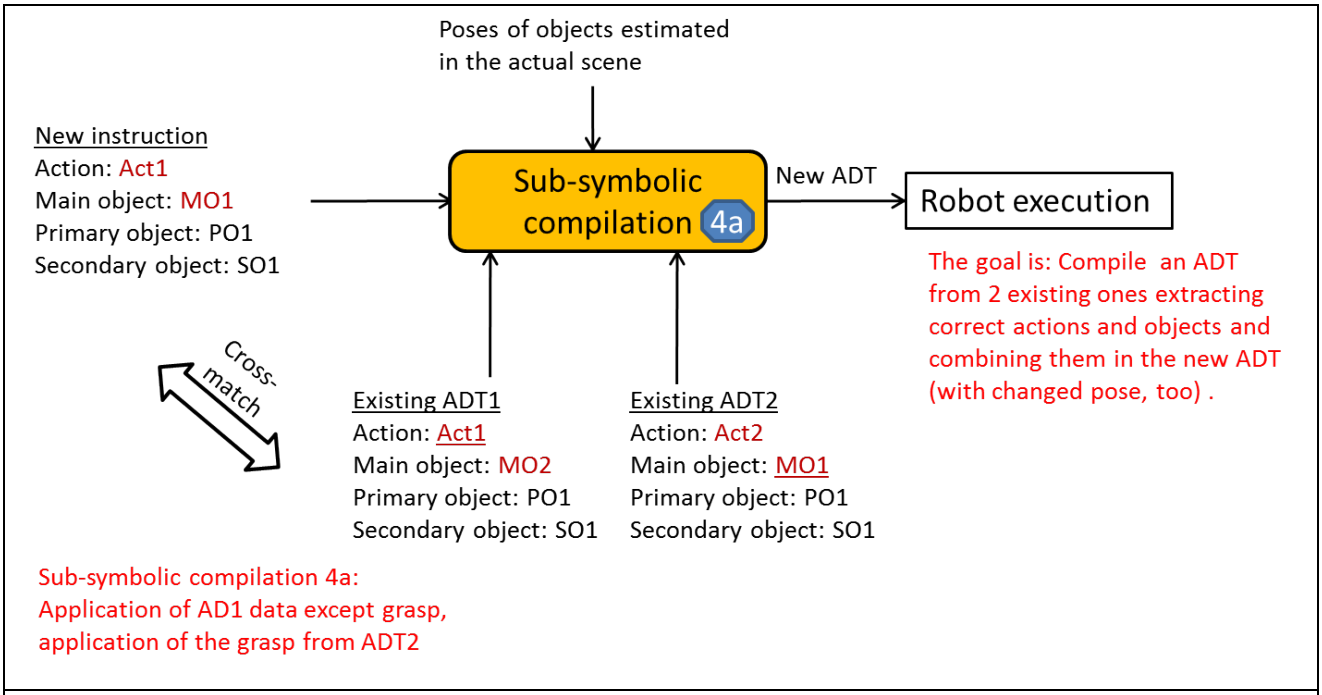

Figure 8. Case 4a: Sub-symbolic compilation of a new ADT in case we have one existing ADT with the required action, and another one with the required main object.

The given instruction is first parsed to determine action type and objects. Then we look for an available ADT with highest similarity with the instruction (ADT1). Since there is one missing object (main object), we look for an ADT which contains the measuring beaker as the main object (ADT2). Then we create a new ADT which is the same as ADT1 except for the main object and its belonging data (approaching pose, grasping). The correspondences of actions and objects are shown in Table 2. The resulting new ADT can then be executed, as will be shown in the demonstrator at the review meeting.

Table 2. Correspondences of action and objects between instruction and existing ADTs, Case 4a.

	action type	main object	primary object	secondary object
Instruction	*PickAndPlace*	*measuring beaker*	*table*	*cup*
ADT1	PickAndPlace	plastic bottle	table	cup
ADT2	Shake	measuring beaker	table	table
New ADT	*PickAndPlace*	*measuring beaker*	*table*	*cup*

In the second case, we have the same instruction again: "Pick the measuring beaker and place it in the cup". The available ADTs, however, are different. The schematic view for compilation is presented in Figure 9. We have two ADTs with the same action type, each containing one of the objects needed in the new instruction. ADT1 is for picking a plastic bottle and placing it in the cup. ADT2 describes picking the measuring beaker and putting it on a plate. In this case we can also perform the instruction, by compilation of a new ADT, where corresponding action and object matches are shown in Table 3.

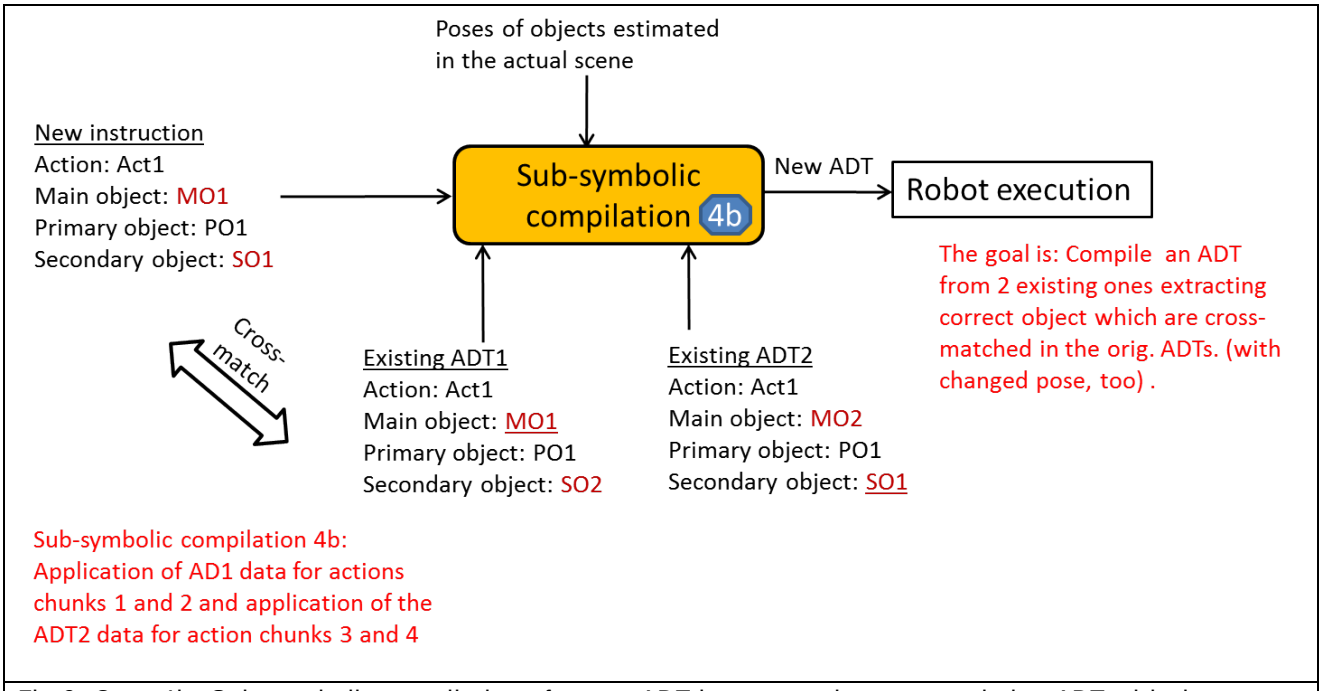

Fig 9. Case 4b: Sub-symbolic compilation of a new ADT in case we have one existing ADT with the required first part of the action, and another one with the required second part of the action.

Table 3. Correspondences of action and objects between instruction and existing ADTs, Case 4b.

	action type	main object	primary object	secondary object
Instruction	*PickAndPlace*	*measuring beaker*	*table*	*cup*
ADT1	PickAndPlace	plastic bottle	table	cup
ADT2	PickAndPlace	measuring beaker	table	plate
New ADT	*PickAndPlace*	*measuring beaker*	*table*	*cup*

Finally, we analyze a situation where we have a set of fully symbolically matching ADTs and we can choose which ADTs fit for re-use best based on numerical parameters. Specifically, we analyze here a “peg in hole” operation where pegs have different length and we re-use those ADTs where peg length is similar to the actual length of the peg provided with the new instruction for generalization towards the new “peg in hole” action. The scheme for this type of sub-symbolic compilation is shown in Figure 10. This type of compilations is achieved by **statistical learning** using the sub-symbolic knowledge, i.e. sensorimotor data stored in rosbags associated with the ADTs. In the end, force feedback is used to ensure correct execution of the operation. This case is part of the IASSES demonstrator.

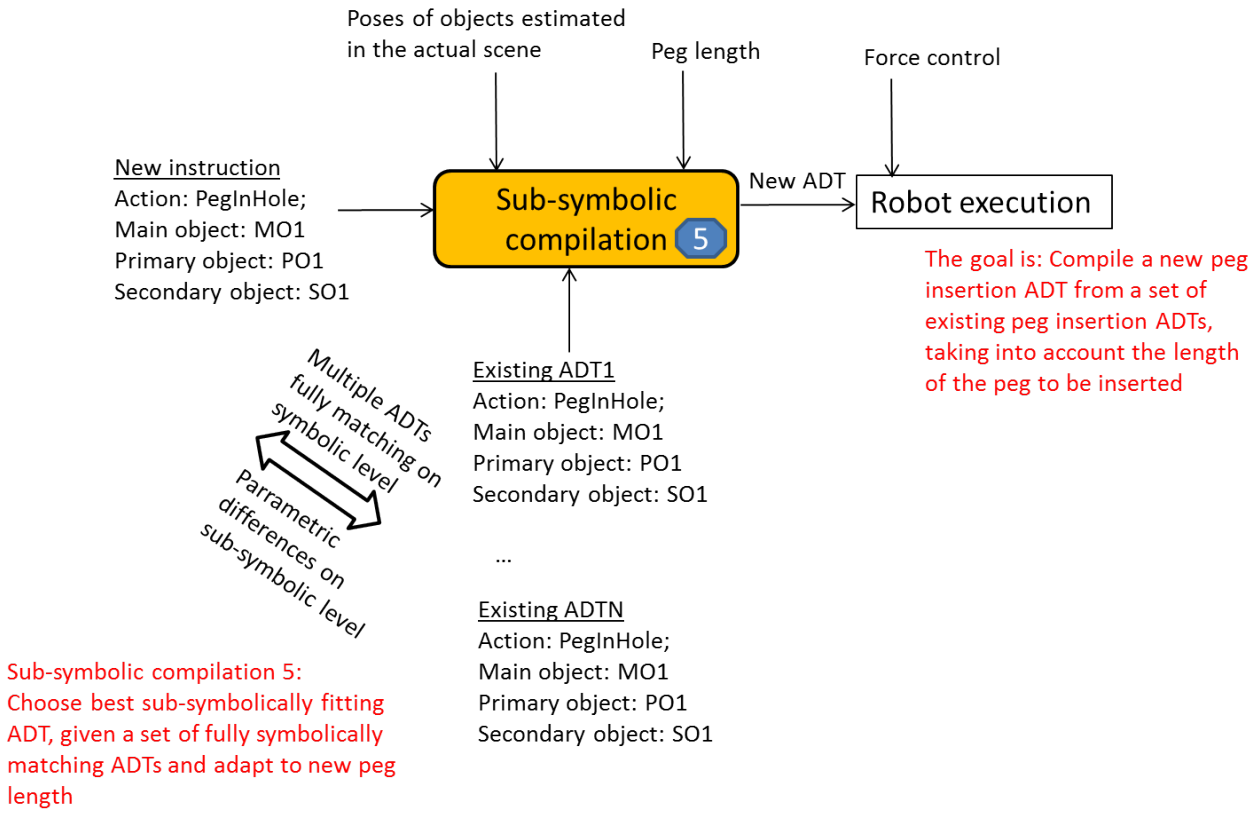

Figure 10. Case 5: Sub-symbolic compilation of a new ADT in case we generalize from several fully symbolically matching ADTs.