Manipulation of Articulated Objects Using Dual-arm Robots via Answer Set Programming

RICCARDO BERTOLUCCI
University of Calabria, Italy
(*e-mail:* bertolucci@mat.unical.it)

ALESSIO CAPITANELLI
Teseo srl, Italy
(*e-mail:* alessio.capitanelli@teseotech.com)

CARMINE DODARO and NICOLA LEONE
University of Calabria, Italy
(*e-mails:* dodaro@mat.unical.it, leone@mat.unical.it)

MARCO MARATEA and FULVIO MASTROGIOVANNI
University of Genoa, Italy
(*e-mails:* marco.maratea@unige.it, fulvio.mastrogiovanni@unige.it)

MAURO VALLATI
University of Huddersfield, UK
(*e-mail:* m.vallati@hud.ac.uk)

Abstract

The manipulation of articulated objects is of primary importance in Robotics and can be considered as one of the most complex manipulation tasks. Traditionally, this problem has been tackled by developing *ad hoc* approaches, which lack flexibility and portability. In this paper, we present a framework based on answer set programming (ASP) for the automated manipulation of articulated objects in a robot control architecture. In particular, ASP is employed for representing the configuration of the articulated object for checking the consistency of such representation in the knowledge base and for generating the sequence of manipulation actions. The framework is exemplified and validated on the Baxter dual-arm manipulator in the first, simple scenario. Then, we extend such scenario to improve the overall setup accuracy and to introduce a few constraints in robot actions execution to enforce their feasibility. The extended scenario entails a high number of possible actions that can be fruitfully combined together. Therefore, we exploit macro actions from automated planning in order to provide more effective plans. We validate the overall framework in the extended scenario, thereby confirming the applicability of ASP also in more realistic Robotics settings and showing the usefulness of macro actions for the robot-based manipulation of articulated objects.

KEYWORDS: answer set programming, robots manipulation, macro actions

1 Introduction

The manipulation of articulated objects plays an important role in real-world robot tasks, both in home and industrial environments (Krüger *et al.* 2009; Heyer 2010). Much attention has been paid to the development of approaches and algorithms for

generating the sequence of movements a robot has to perform in order to manipulate an articulated object. In the literature, the problem of determining the two-dimensional (2D) configuration of articulated or flexible objects has been vastly considered in the past few years (Wakamatsu *et al.* 2006); Capitanelli *et al.* (2017); (Nair *et al.* 2017); Capitanelli *et al.* (2018), whereas the problem of obtaining a target object configuration via manipulation has been explored in robot motion planning (Schulman *et al.* 2013; Yamakawa *et al.* 2013; Bodenhagen *et al.* 2014). A limitation of such manipulation strategies is that they are often crafted specifically for the problem at hand, with the relevant characteristics of the object and robot capabilities being either hard-coded or assumed *a priori* known. Therefore, in these approaches, generalization properties and scalability are somehow limited. In this paper, we present a framework based on answer set programming (ASP) (Niemelä 1999; Baral 2003; Brewka *et al.* 2011) for the automated manipulation of articulated objects in a robot 2D workspace. ASP is a general, prominent knowledge representation and reasoning language with roots in logic programming and non-monotonic reasoning (Gelfond and Lifschitz 1988; 1991). In particular, in this paper, ASP is employed for representing the configuration of the articulated object, for checking the consistency of such representation in the knowledge base, as well as for generating the sequence of manipulation actions, that is, the plan.

The framework is first validated using a Baxter dual-arm manipulator in a *simple scenario*, which involves the manipulation of an articulated object in a 2D workspace with the possibility of performing such actions like rotating one of its link with respect to another one around their joint. Afterward, we extend the simple scenario to allow for a higher accuracy of the mockup, and we introduce some constraints in robot actions execution to enforce their feasibility. Such *extended scenario* entails a high number of actions to be possibly executed in sequence, which can be fruitfully combined. Therefore, we exploit *macro* actions (Chrpa *et al.* 2015; Gerevini *et al.* 2015; Chrpa and Vallati 2019) from automated planning in order to generate more compact and effective plans. Macro actions can be considered as sequences of *elementary* actions that, on an application viewpoint, would be useful to be performed in such a sequence, and be considered as a *single* action. In Robotics applications, macro actions may be useful given that performing a sequence of multiple elementary – yet atomic – actions is expected to be more time-consuming than executing the relative macro. In particular, in our application scenario, we have defined macro actions considering sequences of actions that are *often* performed in sequence by the robot, thus meeting such desirable property. We validate the whole framework in this extended scenario, confirming the applicability of ASP-based knowledge representation and reasoning in more realistic Robotics contexts as well, at the same time showing the usefulness of macro actions in this peculiar application.

This paper is an extended and revised version of a paper appearing in the proceedings of LPNMR'19 (Bertolucci *et al.* 2019). The main improvements of the current paper are (*i*) the definition of a new, more realistic scenario, (*ii*) the extension and validation of the whole framework on this new scenario, and (*iii*) the exploitation of macro actions for the new scenario, including a new Action Planning encoding employing macro actions and related experimental analysis of the module and validation of the framework.

The paper is structured as follows. Section 2 presents the problem statement and the simple scenario. Section 3 shows the overall robot control architecture, while Section 4 details the modules where ASP is employed. The framework validation in the simple

scenario is discussed in Section 5. Section 6 introduces the extended scenario, as well as the needed extensions to the ASP encoding. Section 7 presents the defined macros and how they have been encoded in ASP. The paper ends in Section 8 by presenting the analysis we have done on the extended scenario, by discussing related research in Section 9, and by drawing some conclusions in Section 10.

2 Problem statement and the simple scenario

In this section, we define more in detail the addressed problem and we introduce the simple scenario we consider first.

2.1 Problem statement

Our goal is to present (i) an efficient ASP-based representation, planning and execution architecture for the manipulation of articulated objects in terms of perceptual features, their representation and the planning of manipulation actions, which maximizes the likelihood of being successfully executed by dual-arm robot manipulators, and (ii) given a specific articulated object's goal configuration, determine a plan to attain it, whereby each step involves one or more manipulation actions to be executed by a dual-arm robot. In so doing, our working assumptions are as follows:

A_1 flexible objects can be appropriately modeled as articulated objects with a high number of links and joints, as it is customary (Yamakawa *et al.* 2013);

A_2 an articulated object is manipulated while placed on a table, and therefore its 2D configuration is only affected by robot manipulation actions, whereas the effects of external forces such as gravity are not considered;

A_3 we do not consider possible issues related to grasping or motion dexterity during the manipulation task in reasoning process;

A_4 sensing is affected by noise, but the *symbol grounding problem*, that is, the association between perceptual features and the corresponding symbols (Harnad 1990), is assumed to be solved.

On the basis of assumption A_1, we focus on articulated objects only. We define an articulated object as a pair $\alpha = (\mathcal{L}, \mathcal{J})$, where $\mathcal{L}$ is the ordered set of its $|\mathcal{L}|$ links and $\mathcal{J}$ is the ordered set of its $|\mathcal{J}|$ joints. Each link $l \in \mathcal{L}$ is characterized by two parameters, namely a length λ_l and an orientation θ_l. Considering assumption A_2, we restrict the set of all possible configurations to 2D configurations only. Also, we allow only for a limited number of possible orientations, which induces a finite set of allowed angle values. If α is represented using *absolute* angles (i.e., with respect to an external Cartesian reference frame as shown in Figure 1 on the top), then its configuration is an $|\mathcal{L}|$-tuple:

$$C_{\alpha,a} = \left(\theta_1^a, \ldots, \theta_{|\mathcal{L}|}^a \right). \tag{1}$$

Otherwise, if relative angles are used (e.g., with respect to the previous link, as shown in Figure 1 on the bottom), then the configuration must be *augmented* with an initial *hidden* link l_0 in order to define a reference frame needed for actual manipulation actions:

$$C_{\alpha,r} = \left(\theta_0^r, \theta_1^r, \ldots, \theta_{|\mathcal{L}|}^r \right). \tag{2}$$
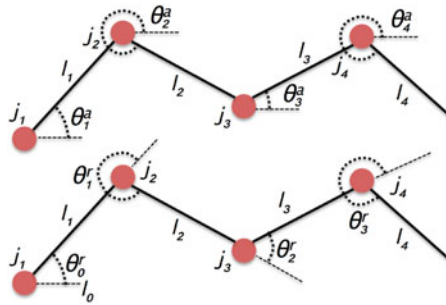
Fig. 1. Two possible object representations: absolute (top) and relative (bottom).

In fact, while in principle the relative angles approach could represent the configuration of an articulated object with one joint less compared with the absolute angles one, the resulting representation would not be unique (actually, there would infinitely many), since the object maintains relative orientations among its parts even when rotated *as a whole*. Obviously enough, this is not a practical way for grounding manipulation actions in the robot workspace, while at the representation level would provide nonetheless a complete representation of the articulated object configuration Capitanelli *et al.* (2018). In this paper, we adopt an absolute angles representation, and we refer to the configuration of an articulated object α simply as C_α.

The problem we want to formalize and solve in this paper can be described as follows. Given an articulated object α, and having defined the number of its $|\mathcal{L}|$ links and $|\mathcal{J}|$ joints, we want to determine an ordered sequence of grounded actions, that is, a plan $\mathcal{P}$, that is,

$$\mathcal{P} = \left\{ a_1, \ldots, a_{|\mathcal{P}|} \right\}, \tag{3}$$

such that its execution by a dual-arm robot manipulator leads from an initial articulated object configuration $C_{\alpha,i}$ to a goal configuration $C_{\alpha,g}$, whereas actions in $\mathcal{P}$ are instances of *a priori* defined planning operators. The set of planning operators differs whether we consider the simple or the extended scenario. While in the simple scenario we model only manipulation actions aimed at re-orienting pairwise links, in the extended scenario we also model link grasping and releasing operators by robot grippers.

2.2 Simple scenario

In order to comply with assumption A_2, which allows us to focus on the manipulation process, we have set up a scenario in which a dual-arm Baxter robot manipulates an articulated object that is conveniently located on a table in front of it.

In the setup, the table sustains the articulated object while it is being manipulated by the robot, and it is assumed to be large enough to accommodate the whole object itself, see Figure 2. As a consequence, link rotations occur only around axes centered on specific object joints, but always perpendicular to the table surface. We have crafted two wooden articulated objects of different size: the first, which is simpler, has three 40-cm long links (which are connected by two in-between joints), whereas the second is made up of five 20-cm long links (connected by four joints). For both objects, links are
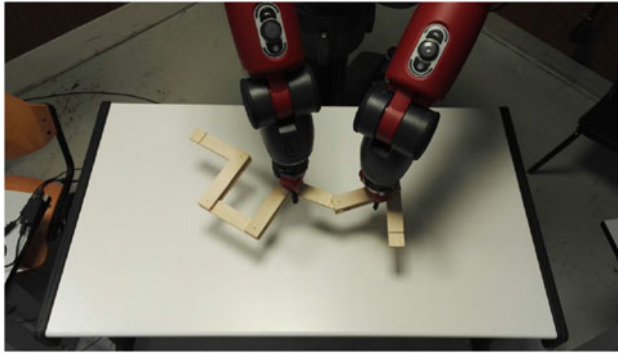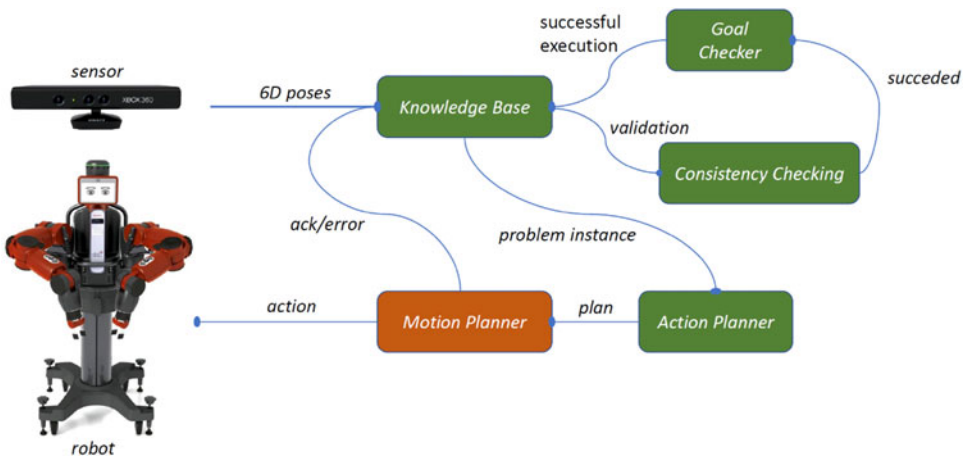
Fig. 2. The experimental scenario.

Fig. 3. The system's architecture: in *green* ASP-based modules, in *orange* the robot-specific control module.

2 cm wide and 3 cm thick. The two objects have been designed to reduce the likelihood of manipulation-specific issues when using the Baxter's standard grippers, in order to comply with assumption A_3. The Baxter's *head* is equipped with a camera pointing downward to the table and able to acquire images of the relevant robot manipulation workspace. QR tags are attached to each object link, which reduces perception errors and is aimed at enforcing assumption A_4. Each QR code provides an overall 6D link pose, which directly maps to an absolute link orientation θ_l^a.

3 System's architecture

Stemming from previous work Capitanelli *et al.* (2017; 2018), the overall system's architecture is a hybrid, reactive/deliberative framework including perceptual, knowledge representation and reasoning, as well as action modules, as shown in Figure 3. Although the target robot platform is the Baxter dual-arm robot manipulator from Rethink Robotics, in principle, the architecture can be adapted to other manipulators as well, either in sim-

ulation or in real-world conditions, as long as appropriate perception, low-level motion planning algorithms, and manipulation strategies are adopted.

In the current implementation, perception modules acquire a video stream using an RGB-D camera sensor located on top of the robot's *head* and pointing downward, which provides 6D poses for each link. In order to simplify and make the perception process more robust, the various links of the articulated object are uniquely identified by fiduciary markers, that is, QR tags. Although more advanced perception strategies are certainly possible, this does not limit the results and the scope of this paper. The perception pipeline generates a set of 6D pose data structures, each one for an object link, which are used to update corresponding ASP-based representation structures in the *Knowledge Base* module. Once such representation structures in *Knowledge Base* are updated, the *Consistency Checking* module verifies that facts within it are not mutually inconsistent, and that numerical values associated with joint angles are within the proper ranges. Relevant parts of the current ASP-based knowledge base are processed to this aim by the rules encoded in the *Consistency Checking* module. As a result, a ASP-based problem instance is generated, whose goal and initial conditions depend on the target articulated object configuration, and the current configuration maintained in the *Knowledge Base* module, respectively. The *Action Planner* module receives such problem instance and generates a plan in the form of an ordered sequence of manipulation actions to be performed by the robot on the articulated object. Once a plan is generated, its actions are processed sequentially to drive the overall behavior of the robot by the *Motion Planner* module, which is responsible for the execution of all manipulation actions. Each action in the simple scenario involves rotating a target link with respect to another, connected one, and in particular around axes centered on the link joints, whose poses are maintained in the *Knowledge Base* module. Any action may be either successful or not, depending on a number of reasons related to perception noise, and grasping or manipulation faults in real-world environments. If a manipulation action is successful, the *Motion Planner* proceeds with the one that follows until the plan ends and the *Knowledge Base* module is notified about successful execution. Otherwise, an issue is raised and re-planning occurs, thereby reiterating the whole workflow described above. The *Goal Checker* is indeed aimed at detecting whether the remaining part of the (already computed) plan can be successfully executed. If not, a re-planning process must occur using the current perceived environment configuration as a new initial configuration.

It is noteworthy that all modules except *Motion Planner* (i.e., all *green* modules in Figure 3) are based on ASP. The *Motion Planner* module encodes all the robot-specific details related to kinematics constraints, dynamics, as well as control. The interested reader is referred to Darvish *et al.* (2018) for a thorough description of the module. One of our architectural assumptions is the separation between the deliberative layer (implemented with ASP encodings) and the robot control layer, although such an assumption is characterized by well-known limitations, discussed in Darvish *et al.* (2018). As such, we aimed at abstracting and simplifying as much as possible all the issues typically related to more realistic scenarios in Robotics, that is, perception, grasping, and manipulation. To this aim, we decided not to consider the articulated object physical properties, such as material or friction, and we assume its perception to be *perfect* since we use QR tags to identify its links. Grasping and manipulation do not employ any feedback-in-the-loop sensing, and unsuccessful actions are recognized only by their effects detected after actions end by the expected location of QR codes.

```
joint(1..5).  angle(0).  angle(90).  angle(180).  angle(270).
isLinked(1,2).  isLinked(2,3).  isLinked(3,4).  isLinked(4,5).
hasAngle(1,90,0).  hasAngle(2,180,0).  hasAngle(3,180,0).
hasAngle(4,270,0).  hasAngle(5,270,0).  time(0..timemax).
goal(1,270).  goal(2,270).  goal(3,180).  goal(4,270).
goal(5,270).  #const granularity = 90.
```

Fig. 4. An example of a knowledge base encoded in ASP for a five-link articulated object.

4 ASP-based modules in the system's architecture

In this section, we describe how ASP is used to implement the modules depicted in Figure 3. In the following, we assume that the reader is familiar with ASP and ASP-Core-2 input language specifications (Calimeri *et al.* 2020).

4.1 Knowledge base

The knowledge base consists of facts over atoms of the form `joint(J)`, `isLinked(J1,J2)`, `angle(A)`, `time(T)`, `hasAngle(J,A,T)`, and `goal(J,A)`, and the constants `timemax` and `granularity`. Atoms over the predicate `joint` represent the joints of the articulated object. Atoms over the predicate `isLinked` represent links between joints `J1` and `J2`. Atoms over the predicate `angle` represent the possible angle values that can be taken by joints, and they can range from 0 to 359. Actually, the atom `angle(0)` must be always part of the knowledge base and admissible angles are the ones that can be obtained by rotating a joint by the degrees specified by the constant `granularity`, for example, if the granularity is 90 degrees, then the admissible angles are 0, 90, 180, and 270. Atoms over the predicate `time` represent the possible time steps, and they range from 0, which represents the initial state, to `timemax`. Atoms over the predicate `hasAngle` represent the angle `A` of the joint `J` at time `T`. The knowledge base only contains the initial state of each joint, that is, its angle at time 0. Finally, atoms over the predicate `goal` represent the angle `A` that must be reached by the joint `J` at the time step specified by `timemax`. An example of the input is represented by the facts and constants reported in Figure 4. It is noteworthy that the constant `timemax` is not included in the example, and its usage will be described in Section 4.3.

4.2 Consistency checking

The *Consistency Checking* module verifies the mutual consistency of the facts maintained within the knowledge base, in order to assure that the later Action Planning does not draw conclusions on inconsistent pairwise facts.

This is done using the ASP encoding reported in Figure 5. In particular, rules c_{1a} and c_{1b} check whether atoms over the predicate `isLinked` represent the links between two joints, while c_2 checks whether there is no link between the same joint. Rules c_{3a}, c_{3b}, and c_{3c} check whether the predicate `hasAngle` expresses angles in the proper format, whereas c_{4a} and c_{4b} check the correctness of the predicate `goal`. Rules c_5 and c_6 check whether at most one goal is specified for each joint (in order to avoid multiple goals for the same joint which would confuse the reasoner), whereas rules c_7 and c_8 verify whether each joint is in exactly one angle at time step 0 (for the same reason). Rules c_9 and c_{10}

```
c₁ₐ    :- isLinked(J1,J2), not joint(J1).
c₁ᵦ    :- isLinked(J1,J2), not joint(J2).
c₂     :- isLinked(J,J).
c₃ₐ    :- hasAngle(J,A,T), not joint(J).
c₃ᵦ    :- hasAngle(J,A,T), not angle(A).
c₃c    :- hasAngle(J,A,T), not time(T).
c₄ₐ    :- goal(J,A), not joint(J).
c₄ᵦ    :- goal(J,A), not angle(A).
c₅     moreThanOneGoal(J) :- joint(J), #count{A:goal(J,A)}>1.
c₆     :- joint(J), moreThanOneGoal(J).
c₇     oneStartingAngle(J) :- joint(J), #count{A:hasAngle(J,A,0)}=1.
c₈     :- joint(J), not oneStartingAngle(J).
c₉     :- not time(0).
c₁₀    :- not angle(0).
c₁₁    possibleAngle(0).
c₁₂    possibleAngle(X) :- possibleAngle(Y), X=(Y+granularity)\360.
c₁₃    :- not angle(X), possibleAngle(X).
c₁₄    :- angle(X), not possibleAngle(X).
```

Fig. 5. ASP encoding for consistency checking.

simply verify the existence of the first time step and angle 0, respectively. Finally, rules from c_{11} to c_{14} check whether atoms over the predicate `angle` represent allowed angles.

4.3 Action Planning

ASP is not a planning-specific language, but it can be also used to specify encodings for planning domains (Lifschitz 2002), like the problem we consider in this paper. We have defined several encoding variants, for what concerns either the manipulation modes and the strategy for computing plans. The encoding described in this section is our base encoding, and it is embedded into a classical iterative deepening approach in the spirit of SAT-based planning Kautz and Selman (1992), where `timemax` is initially set to 1 and then increased by 1 if a plan is not found, which guarantees to generate the shortest possible plans for a sequential encoding, that is, when the robot performs only one action for each step (see Section 9 for more details about other strategies).

Figure 6 reports the base encoding. It is noteworthy that it uses operations $\backslash$ and $|\cdots|$, which are not defined in the ASP-Core-2 standard but supported by Clingo (Gebser *et al.* 2016) that compute the remainder of the division and the absolute value, respectively. This encoding allows for forward propagation only, that is, manipulation actions *in one direction* along the articulated object chain, for example, it is possible to move the second joint holding the first joint but not the opposite.

Since we employ an absolute representation, r_1, r_2, and r_3 add to the knowledge base the `joint(0)`, its angle, and the link to joint 1. This joint is not meant at being moved, and it is used only to have a fixed reference between the robot-centered and the articulated object frames. Rule r_4 enforces bidirectionality of linked joints, that is, if `joint(1)` is linked to `joint(2)` then `joint(2)` is also linked to `joint(1)`. Then, rule r_5 is used to select an atom of the form `changeAngle(J1,J2,A,Ai,T)`, where `J1` is the joint around which rotation occurs, `J2` is the joint related to the link to be kept steady, `A` is the desired angle, `Ai` is the current angle of `J1`, and `T` is the current step. Rule r_{10} ensures

```
r₁     joint(0).
r₂     hasAngle(0,0,0).
r₃     isLinked(0,1).
r₄     isLinked(J1,J2) :- isLinked(J2,J1).
r₅     {changeAngle(J1,J2,A,Ai,T) : joint(J1), joint(J2), J1>J2, angle(A),
            hasAngle(J1,Ai,T), A<>Ai, isLinked(J1,J2)} <= 1
                 :- time(T), T < timemax, T > 0.
r₆     ok(J1,J2,A,Ai,T) :- changeAngle(J1,J2,A,Ai,T),
            F1=(A+granularity)\360, F2=(Ai\360), F1=F2, A < Ai.
r₇     ok(J1,J2,A,Ai,T) :- changeAngle(J1,J2,A,Ai,T),
            F1=(Ai+granularity)\360, F2=(A\360), F1=F2, A > Ai.
r₈     ok(J1,J2,A,0,T) :- changeAngle(J1,J2,A,0,T), A=360-granularity.
r₉     ok(J1,J2,0,A,T) :- changeAngle(J1,J2,0,A,T), A=360-granularity.
r₁₀    :- changeAngle(J1,J2,A,Ai,T), not ok(J1,J2,A,Ai,T).
r₁₁    affected(J1,An,Ac,T) :- changeAngle(J2,_,A,Ap,T), hasAngle(J1,Ac,T),
            J1>J2, angle(An), An=|(Ac + (A-Ap)) + 360|\360, time(T).
r₁₂    hasAngle(J1,A,T+1) :- changeAngle(J1,_,A,_,T).
r₁₃    hasAngle(J1,A,T+1) :- affected(J1,A,_,T).
r₁₄    hasAngle(J1,A,T+1) :- hasAngle(J1,A,T), not changeAngle(J1,_,_,_,T),
            not affected(J1,_,_,T), T <= timemax.
r₁₅    :- goal(J,A), not hasAngle(J,A,timemax).
```

Fig. 6. Encoding for the simple scenario: it allows for forward manipulation actions only.

the validity of the configuration represented by the atom `changeAngle(J1,J2,A,Ai,T)`, that is when each action has a desired angle `A` that can be reached in one step (rules r_6, r_7, r_8, and r_9). Rule r_{10} takes advantage of the atoms over the predicate `ok`, which are used to model the valid configurations of the atom `changeAngle(J1,J2,A,Ai,T)`. Rule r_{11} is used to identify which joints are affected by the atom selected in r_5. In particular, atoms of the form `affected(J1,An,Ac,T)` represent the fact that, at time step `T`, the angle associated with joint `J1` is updated from `Ac` to `An`. Rules r_{12} and r_{13} are used to update the joints' angle values for the next step, while r_{14} states that if neither r_{12} nor r_{13} have affected any joint then the corresponding angle values remain unchanged. Finally, r_{15} states the the goal must be reached.

It is noteworthy that the encodings presented in this paper assume that the resulting (intermediate) configurations of the articulated object are feasible in practice, for example, in case of link overlaps. While it could be certainly possible to reason about the pairwise mutual position and orientation of each link, that is definitely out of the scope of this work. It must be noted, however, that such assumption is quite common in hybrid reactive/deliberate robot control architectures, although a few works in the literature explore the interplay between the reactive and the deliberative layers (Thomas *et al.* 2018; 2019).

4.4 Goal checker

During the execution of a plan, it may happen that due to errors in robot manipulation actions, or as a result of human interventions in human–robot cooperation scenarios Capitanelli *et al.* (2018), actual joint angle values (and therefore the relative displacement of links) may be different with respect to those represented in the knowledge base, which are propagated forward assuming that proper action execution.

If this happens, it is necessary to check whether the current configuration of the articulated object is still compatible with the plan as it is being executed by the robot. While small variations could be easy to accommodate, and not requiring re-planning should the symbolic representation be preserved notwithstanding the differences in numerical values, big variations may lead to qualitatively different configurations for the articulated object, thereby the computation of a new plan from a new configuration may be needed. This can be accomplished by asynchronously creating a new initial object configuration based on the current robot perception about its workspace and the articulated object in particular. In this process, the role of the *Goal Checker* module is to check whether there is the need to create such a new object configuration, that is, when the intermediate goal has not been reached for the reasons outlined above. This is done using rule r_{15} from the encoding in Figure 6.

5 Simple scenario and the first experimental validation

In order to perform the first validation of the whole architecture, we have prepared a setup whereby a dual-arm Baxter robot has to manipulate a five-link articulated object. The object is located on a table in front of the robot. Articulated objects made up of five links provide a very valuable ground for testing our approach, as the *limited* number of links is expected not to make the manipulation difficult for the robot, whereas such number proves to be sufficiently high to require the development of complex plans to reach a goal configuration. The use of a dual-arm manipulator like Baxter is justified by its widespread adoption as a research platform, and by the necessity to employ a robot with two arms in order to manipulate the object, that is, the robot must hold one link of the object while rotating an adjacent one. It may be argued that robot simulations may be characterized by some practical advantages in this scenario. Indeed, they would allow us to run a greater number of planning execution cycles with minimal human supervision, shorter execution times, and a reduced number of errors in perception or action. Moreover, simulations are less susceptible to uncertainty and low-level motion planning failures, which are outside the scope of this work. Nonetheless, we preferred to test our architecture with a real robot platform in order to provide a more robust proof of concept of the approach. As a matter of fact, such choice and the related experiences led to the development of the extended scenario, which is described in the next section. For these reasons, we provide an example along with a discussion about the limitations of the scenario, and we defer to the extended scenario for a more thorough analysis of our architecture.

Currently, the developed software architecture adopts both off-the-shelf and custom modules. In particular, we employed ALVAR, a QR tags tracking library typically used in augmented reality applications, to detect the absolute pose of articulated object links using a camera, as well as MoveIt!, which is a *de facto* standard for motion planning and execution in the Robotics community. Modules related to ASP have been developed custom. The architecture has been designed and implemented using the Robot Operating System (ROS, Indigo release) framework, and runs on a machine with an Intel i7-4790 CPU and 16 GB of RAM.

Tests have been carried out as follows. First, the articulated object is located on the table in front of the robot in a *random* configuration that is compatible with the link

a_1 : changeAngle(2,1,90,180,1) a_2 : changeAngle(1,0,180,90,2)
a_3 : changeAngle(3,2,90,180,3) a_4 : changeAngle(1,0,270,180,4)

Fig. 7. An excerpt of the answer set returned by Clingo for a problem grounded in the simple scenario. Labels $a_1 \dots a_4$ are compact references for ground actions.

orientation granularity encoded in the knowledge base, and within an acceptable perceptual error margin. A video stream is acquired by the camera, QR tag poses are extracted by ALVAR, and the *Knowledge Base* module updated accordingly. It is noteworthy that images are acquired at a 10-Hz frequency, whereas the knowledge base is updated at a 1-Hz frequency. Such a lower frequency update is due to the preprocessing of QR tag poses, which are stabilized using state estimation and filtering techniques. Initial and goal configurations, as well as information about the length, position and orientation of each link, are therefore represented in terms of the ASP atoms described in Section 4.1.

As described above, the *Goal Checker* module continuously control whether the information provided by the perception layer is compatible with the desired articulated object expected configurations. When this happens, the *Knowledge Base* module is notified, and plan execution proceeds as expected until the goal configuration is reached. In this case, the *Knowledge Base* module does not produce a new problem instance for the *Action Planning* module, and therefore execution stops. Otherwise, if the *Goal Checker* module finds an incoherence in the knowledge representation structure, for example, the currently perceived articulated object configuration is not compatible with the expected one, then execution is interrupted, and the *Knowledge Base* module updates the problem instance for the *Action Planning* module accordingly. Such inconsistencies may typically occur in two cases, both generated by rules c_{12}, c_{13}, and c_{14} shown in Figure 5. As an example, we can consider human–robot cooperation scenarios, whereby a human operator and a collaborative robot may interact to jointly manipulate the articulated object Capitanelli *et al.* (2018). On the one hand, if the robot were loosing grip on a link while manipulating it, or the human operator were rotating a link to an orientation not compatible with the set of represented ones, a consistency exception would rise. On the other hand, if the unexpected orientation were compatible with the set of represented ones, plan execution would proceed flawlessly until the robot had to perform an action on that link. In such a case, the *Motion Planning* module would expect a certain link orientation different from the current one, which would raise a consistency exception and cause re-planning.

On the basis of the encoding described in Section 4.3, the *Action Planning* module exploits the state-of-the-art ASP solver Clingo (Gebser *et al.* 2016) to generate a (valid) plan. It is important to notice that the video stream from the camera is continuously acquired, and therefore the *Knowledge Base* module updated, and *Consistency Checking* performed. In order to avoid false-positive consistency exceptions, for example during robot manipulation actions, the update of the knowledge base is paused during their execution.

An example of the whole process is shown in Figures 4, 7, and 8. Figure 4 reports an example of ASP-based representation of a problem instance in the simple scenario, whereby the number of joints of the articulated object, their initial poses, and the goal configuration to achieve are specified. Figure 7 lists an excerpt of an answer set obtained by Clingo with the encoding presented in Section 4.3. Each atom of the form changeAngle in the answer set represents an action to perform on a joint, with the meaning detailed in
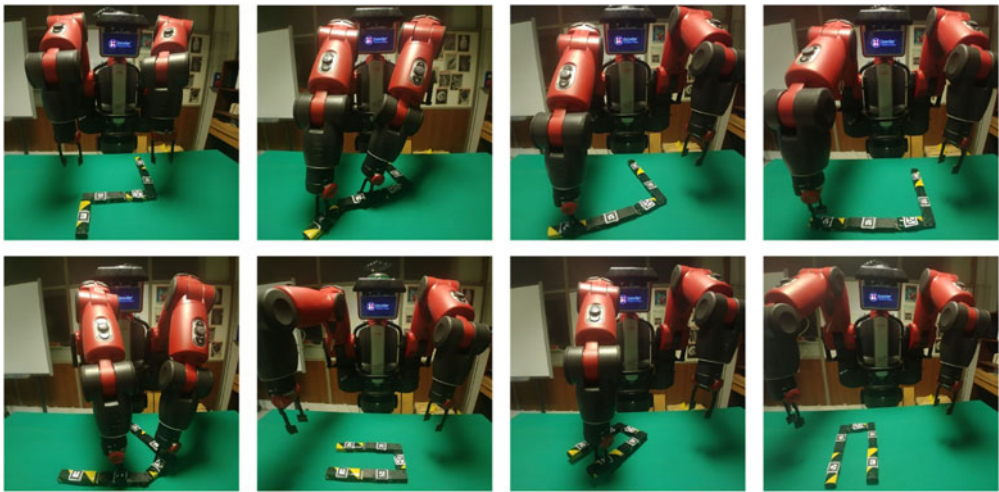
Fig. 8. Execution of a problem instance in the simple scenario.

Section 4.1. Figure 8 provides a few snapshots associated with the plan execution process, from top-left to bottom-right. In particular, starting from the initial configuration of the articulated object shown in the top-left Figure, the second, fourth, fifth, and seventh figures represent the execution of actions a_1, a_2, a_3, and a_4, respectively, as shown in Figure 7, whereas the third and the sixth represent intermediate configurations. Finally, the bottom-right figure shows the final state corresponding to the articulated object goal configuration. It is important to note that the fourth figure displays both a_3 execution and its resulting intermediate state, since it just corresponds to a rotation of the whole object.

As part of this initial experimental assessment, we performed also a computational analysis on the performance associated with the *Action Planning* module, by varying the number of links (up to 12), the angle granularity, and by randomly generating initial and final configurations, for a total of 280 instances. On the successfully solved problem instances, Clingo took a 1.5-s processing time on average and could solve the problems in around eight steps on average, with results as low as $0.01\,\text{s}/4$ steps and never above $2.2\,\text{s}/9$ steps, which confirms the applicability of ASP reasoning in this scenario. All the plans have been validated with the VAL tool (Howey *et al.* 2004).

Albeit the computational performance deteriorates when both the number of links and allowed orientations are increased, results are encouraging considering the limited robot workspace size, and the robot dexterity capabilities for grasping and manipulation, which represent the real drawback of the simple scenario. As a matter of fact, the majority of run-time issues are related to the impossibility for the robot to reach with both arms the required grasping poses on the articulated object links due to the fact that rotations *tend* to displace the whole object on the table to a great extent. It is noteworthy, however, that the proposed ASP-based approach is guaranteed to compute the shortest plan leading to the target object configuration due to the use of an iterative deepening procedure. This is pivotal, as it allows minimizing the actual execution time of the plan, which is the most time-consuming part of the whole process.

6 Extended scenario and the updated encoding

Analyzing the overall execution performance of our framework in the simple scenario, we noticed that the robot was not always able to perform the required manipulation actions due to the unreachability of the grasping poses associated with the links to manipulate, for example, for links displaced outside the robot manipulation space.

These considerations led us to the design of a new, extended scenario. The scenario does not modify any physical characteristics with respect to the setup introduced in Section 2.2 but represents the setup and its constraints with a higher accuracy with the aim of enforcing actions feasibility. In particular, the extended scenario builds up on a model with an improved trade-off between accurately modeling the physics of the specific scenario and having a general model that can be easily reused in similar settings. To this aim, the most sensible aspect of the scenario at hand is related to links manipulation, so in the new encoding it is modeled with a greater level of detail. Furthermore, there are different ways whereby a robot can center a link to manipulate with respect to its manipulation space. For this reason, the newly introduced encoding provides a more general way for modeling this specific aspect. On top of these considerations, there is still the need for the model to be operational, so that ASP solvers can find solutions in a reasonable amount of time.

In this section, we present, in two separate subsections, the extended scenario we have designed, and the related, improved encoding we have defined in order to deal with such scenario in our robot control architecture.

6.1 Extended scenario

Herewith, we briefly describe the main differences with respect to the simple scenario. These can be summarized as follows.

- *Robot grippers are explicitly modeled.* Each robot gripper should be considered as a resource that can be either *occupied*, for example, keeping a link firmly or rotating a link, or *free*. With this modification, it is possible to explicitly represent which robot gripper can manipulate a given link.
- In the simple scenario, it may happen that a link could not be reached or grasped or it may be placed to an area of the robot's workspace where manipulation could be difficult or even impossible because of the robot kinematics configuration. This situation may happen for reasons related to the articulated object's configuration while being manipulated, or the specific sequence of manipulation actions required by the plan. In the extended scenario, each time a manipulation action is carried out on a given link, *it is assured that the target link is centered in the robot workspace.* If this is not the case, the link is first moved toward the central part of the table. This maximizes the likelihood of a relevant link to be successfully reached and manipulated by the robot.
- *Grasping and release actions by the two grippers are explicitly modeled.* In the scenario described in Section 2.2, these two manipulation actions were not modeled, although they were assumed to be properly carried out during the execution phase, as part of each action execution. By considering an explicit modeling, we can represent grippers occupancy, and we can better characterize the semantics associated with each action, since now grasping, manipulation, and release are distinct.

It is noteworthy that the above-mentioned features allow for a number of improvements. We can envisage two main advantages. On the one hand, this encoding is expected to better manage the explicitly modeled robot resources, that is, the grippers. On the other hand, manipulation actions are now characterized by a more precise semantics, which does not make any implicit assumption about actual robot behavior. In a nutshell, the encoding provides a more accurate description of the domain, that supports a less abstract planning process, and can easily generalize to different conditions (e.g, by modeling robot resources, you can also deal with cases where some of those resources are unavailable).

Furthermore, in the new encoding, we provide a specific action for displacing links to the center of the robot manipulation space, which enforces the reasoner to take positioning into account, but does not explicitly mention the sequence of movements to be done for this operation. Such choice has been taken for two main reasons: (*i*) different robots may employ different strategies to displace the object, for example, using a combination of manipulation and pushing behaviors, so we aimed at providing a model that can be easily reused with other platforms and (*ii*) the way a link is moved to the center of the workspace requires a somewhat lower degree of accuracy when compared to a manipulation action.

6.2 ASP encoding for the extended scenario

In this section, we describe ASP-related modifications to the *Knowledge Base* module and the *Action Planning* module to implement the extended scenario.

ASP encoding for the Knowledge Base module. In order to have a working architecture in the extended scenario, we have to update the *Knowledge Base* module, which has been extended to include more atoms. Atoms `gripper(1)` and `gripper(2)` represent the two robot grippers. Atoms `free(1,0)` and `free(2,0)` represent the fact that at time step 0, the two robot grippers are free and ready to grasp. Atom `in_centre(J,0)` represents the fact that the joint `J` is at the center of the workspace at time step 0. Moreover, this last change led us to the necessity to add to the knowledge base both links and joints, because the robot must act on the links but we want that a certain joint is in the middle of the workspace. For this reason, the atoms of the form `isLinked(L1,L2)`, where `L1` and `L2` are two links, became `connected(J,L)`, where `J` is a joint and `L` a link. We also use facts over atoms of the form `link(L)` to represent the links of the articulated object. Finally, we mention that atoms of the form `hasAngle(J,A,T)`, where the first term `J` represents a joint, are replaced by atoms of the form `hasAngle(L,A,T)`, where the first term `L` is a link.

ASP encoding for the Consistency Checking module. The changes to the *Knowledge Base* module require also to update the *Consistency Checking* module in order to keep consistency with the other modules of the architecture. For this reason, we add a check on each new added atom and modified rules that were not needed for the new knowledge base. Such checks are similar to the ones presented in Figure 5 and reported as follows:

```
r5'   {changeAngle(L1,L2,J1,A1,A2,G1,G2,T)} :- link(L1), link(L2),
          joint(J1), angles(A1), angles(A2), gripper(G1), gripper(G2),
          time(T), in_centre(J1,T), grasped(G1,L1,T), grasped(G2,L2,T),
          in_hand(L1,T), in_hand(L2,T), not free(G1,T),
          not free(G2,T), connected(J1,L1), connected(J1,L2),
          hasAngle(L1,A2,T), L1<>L2, G1<>G2
```

Fig. 9. Rule r_5 with its preconditions updated.

```
c3a'    :- hasAngle(L,A,T), not link(L).
c15     :- connected(J,L), not joint(J).
c16     :- connected(J,L), not link(L).
c17     :- in_centre(J,T), not joint(J).
c18     :- in_centre(J,T), not time(T).
c19     :- in_hand(L,T), not link(L).
c20     :- in_hand(L,T), not time(T).
c21     :- grasped(G,L,T), not gripper(G).
c22     :- grasped(G,L,T), not link(L).
c23     :- grasped(G,L,T), not time(T).
c24     :- free(G,T), not gripper(G).
c25     :- free(G,T), not time(T).
```

Note that c_{1a}, c_{1b}, and c_2 are removed since `isLinked(L1,L2)` are not part of the Knowledge Base, whereas c_3 is replaced with $c_{3a'}$.

ASP encoding for the Action Planning Module. Differently from the encoding, we designed for the simple scenario (Figure 6), and which from now on we refer to as *Simple Action Scenario* (SAS), in the encoding for the extended scenario, referred to as *Simple Action Extended Scenario* (SAES) from now on, we added the possibility of performing manipulation actions in both directions, that is, it is now possible to rotate a link with respect to any of its neighboring links in the articulated object chain. This is accomplished by slightly modifying the encoding to allow for both forward and backward propagation. In particular, we removed the constraint $J1 > J2$ in rule r_5 and we changed its body. Such updates are necessary to comply with the new model: since two links must be grasped, we had to consider the presence of the grippers and of the links. Moreover, those links must be at the center of the robot manipulation space and, therefore, it is necessary to ensure that the rule is selected only if the joint between them is located at the center. For the sake of readability we will, from now on, refer to the modified version of rule r_5 as rule $r_{5'}$, as shown in Figure 9. It is important to emphasize here that, aside from such changes, the idea behind the SAS encoding is maintained: the general structure and the strategy employed by the SAES encoding to deal with manipulation actions related to the articulated object configuration changes are the same as in Figure 6.

Figure 10 reports the additional rules needed to deal with the extended scenario. Rules r_{16}, r_{18}, and r_{23} are related to the selection of possible actions in this model: r_{16} locates the joint that has to be moved to the center of the manipulation space, r_{18} selects an atom of the form `take_links_to_move (L1,L2,J,G1,G2,T)`, where `L1` and `L2` are the links that have to be grasped, while r_{23} selects an atom of the form `release_links(L1,L2,J,G1,G2,T)` to release the links the robot is acting upon, respec-

r_{16} `{move_link_to_central(L1,J1,G2,T)}:-link(L1), joint(J1),`
`        gripper(G2), time(T), connected(J1,L1), free(G2,T),`
`        not in_hand(L1,T), not in_centre(J1,T).`

r_{17} `in_centre(J1,T+1):-move_link_to_central(_,J1,_,T), T<timemax+1.`

r_{18} `{take_links_to_move(L1,L2,J1,G1,G2,T)}:-link(L1), link(L2),`
`        joint(J1), gripper(G1), gripper(G2), in_centre(J1,T)`
`        free(G1,T), free(G2,T), not in_hand(L1,T), not in_hand(L2,T),`
`        connected(J1,L1), connected(J1,L2), L1<>L2, G1<>G2.`

r_{19} `in_hand(L1,T+1):-take_links_to_move(L1,_,_,_,_,T), T<timemax+1.`

r_{20} `in_hand(L2,T+1):-take_links_to_move(_,L2,_,_,_,T), T<timemax+1.`

r_{21} `grasped(G1,L1,T+1):-take_links_to_move(L1,_,_,G1,_,T), T<timemax+1.`

r_{22} `grasped(G2,L2,T+1):-take_links_to_move(_,L2,_,_,G2,T), T<timemax+1.`

r_{23} `{release_links(L1,L2,J1,G1,G2,T)}:-link(L1), link(L2), joint(J1),`
`        gripper(G1), gripper(G2), time(T), grasped(G2,L2,T),`
`        grasped(G1,L1,T), in_hand(L1,T), in_hand(L2,T),`
`        not free(G1,T), not free(G2,T), connected(J1,L1),`
`        connected(J1,L2), L1<>L2,G1<>G2.`

r_{24} `free(G2,T+1):-release_links(_,_,_,_,G2,T),T<timemax+1.`

r_{25} `free(G1,T+1):-release_links(_,_,_,G1,_,T),T<timemax+1.`

r_{26} `action(T,move_link_to_central(L1,J1,G2,T)) :-`
`        move_link_to_central(L1,J1,G2,T).`

r_{27} `action(T,take_links_to_move(L1,L2,J,G1,G2,T)) :-`
`        take_links_to_move(L1,L2,J,G1,G2,T).`

r_{28} `action(T,changeAngle(L1,L2,J,A1,A2,G1,G2,T)) :-`
`        changeAngle(L1,L2,J,A1,A2,G1,G2,T).`

r_{29} `action(T,release_links(L1,L2,J1,G1,G2,T)) :-`
`        release_links(L1,L2,J1,G1,G2,T).`

r_{30} `:- time(T), #count{Z : action(T,Z)} != 1.`

r_{31} `in_hand(L,T+1):-in_hand(L,T), not release_links(L,_,_,_,_,T),`
`        not release_links(_,L,_,_,_,T), T<timemax+1.`

r_{32} `free(G,T+1):-free(G,T), not take_links_to_move(_,_,_,_,G,T),`
`        not take_links_to_move(_,_,_,G,_,T), T<timemax+1.`

r_{33} `grasped(G,L,T+1):-grasped(G,L,T), T < timemax+1`
`        not release_links(L,_,_,G,_,T), not release_links(_,L,_,_,G,T).`

Fig. 10. Simple Action Extended Scenario (SAES) encoding.

tively. Rule r_{17} is used to signal which link is located at the center of the space, as effects of `move_link_to_central(L1,J1,G2,T)`. To this end, we use atoms of the form `in_centre(J,T)`, which represent the fact that a joint J is at the center at time step T. Rules r_{19} and r_{20} are used to identify grasped links. In particular, they use atoms of the form `in_hand(L,T)`, representing the fact that a link L has been grasped by one robot gripper at time step T. In a similar way, r_{21} and r_{22} are used to identify which gripper is occupied or is free. To this aim, atoms of the form `grasped(G,L,T)` are used, and they represent the fact that a link L is grasped at the time step T using gripper G. Rules r_{24} and r_{25} are used to notify that robot grippers are free and they can be used again. We used atoms of the form `free(G,T)`, which model the fact that a gripper G is free at time step T. Furthermore, rules from r_{26} to r_{30} ensure that only one action (among the ones represented by rules r_{16}, r_{18}, r_{23}, and $r_{5'}$) is selected for each time step. Atoms of the form `action(T,A)` are used to ensure that action A is executed at time step T. Finally,

the rules from r_{31} to r_{33} are used to propagate the information that has not been changed in the current time step to the next one. It is noteworthy that grasping for *manipulating* links or grasping for *displacing* a link to the center of the manipulation space are very different actions. Links must be grasped in different ways as the robot has to move the arms differently. For this reason, our encoding requires that after the link to be moved has been put in the center of the workspace (rule r_{16}), appropriate grasp actions have to be performed before actual manipulation. Overall, the SAES encoding is composed by:

- the rules from r_1 to r_4 and from r_6 to r_{15} of *SAS* (shown in Figure 6),
- the rule $r_{5'}$ shown in Figure 9, and
- the rules from r_{16} to r_{33} shown in Figure 10.

The rule involving the goal is already present in the *SAS* encoding, that is, r_{15}.

7 Macro actions and their usage in the extended scenario

This section shows how the concept of macro actions, henceforth, possibly referred to simply as *macros*, originally defined in the Automated Planning community, can be fruitfully employed in the context of the extended scenario. The section describes the general concept of what a macro is, how macros can be represented in ASP, and a description about the macros we have employed and their encoding, respectively.

7.1 Macros in automated planning

Macros encapsulate sequences of elementary planning operators. Advantageously, they can be encoded in the same form as planning operators, and therefore they can be added into a domain model seamlessly and can be exploited in a *solver-independent* way. Macros can be seen as *shortcuts* in the state space of a planning problem. They can reduce the number of steps needed to reach the goal state, however, at the cost of an increased branching factor. It is noteworthy that macros can provide a mean for encoding additional, practical, and common sense knowledge about a specific domain. A domain expert can suggest actions that are usually executed in sequence to be encoded as a macro and added to the domain model. On the one hand, if macros are added to the original model, and the related elementary operators are not removed, the extended model is able to cope with cases where encapsulated operators are *usually* executed in sequence, but not always. On the other hand, removing the original operators can lead to more significant performance improvements. It is noteworthy that macros could be considered as a sort of *extra-logic* trick to force a planning engine toward certain trajectories in the planning search space. As such, one may argue whether there could be a better option to implement that trick, for example, in the action execution phase of our architecture. While it may be certainly possible to achieve that, for instance, by performing a post-processing step on a plan made up of elementary actions that would make the architecture less flexible and general-purpose. Instead, macro encodings can be seamlessly integrated with planning operators, and therefore the engineering effort to add, remove, or modify them is minimum.

For the sake of the discussion carried out in the section, let us provide a more formal description of planning operators, and of macros in automated planning. We say that an operator:

$$o = (name(o), pre(o), del(o), add(o)), \tag{4}$$

is a *planning operator*, where $name(o) = op\_name(x_1, \ldots, x_k)$, with op_name is a unique operator name and $x_1, \ldots x_k$ are k variable symbols, that is, arguments, appearing in the operator, and $pre(o)$, $del(o)$, and $add(o)$ are sets of (non) grounded predicates with variables taken only from $x_1, \ldots, x_k$ representing operator precondition, delete, and add effects, respectively. Actions are *grounded instances* of planning operators. An action a is *applicable* in a state s if and only if $pre(a) \subseteq s$. The application of a in s, if possible, results in a state $(s \setminus del(a)) \cup add(a)$; we call $(del(a)) \cup add(a)$ `effects`. Therefore, a macro $o_{i,j}$ is constructed by orderly assembling planning operators o_i and o_j as follows. Let Φ and Ψ be mappings between variable symbols, because we may need to appropriately rename variable symbols of o_i and o_j to construct $o_{i,j}$, the macro is constructed as follows:

- $pre(o_{i,j}) = pre(\Phi(o_i)) \cup (pre(\Psi(o_j)) \setminus add(\Phi(o_i)))$,
- $del(o_{i,j}) = (del(\Phi(o_i)) \setminus add(\Psi(o_j))) \cup del(\Psi(o_j))$,
- $add(o_{i,j}) = (add(\Phi(o_i)) \setminus del(\Psi(o_j))) \cup add(\Psi(o_j))$.

Longer macros, that is, macros encapsulating longer sequences of original planning operators, can be constructed iteratively using the same approach.

For a macro to be *sound*, no instance of $\Phi(o_i)$ can delete an atom required by a corresponding instance of $\Psi(o_j)$, otherwise they cannot be applied consecutively, whereas it is obvious that if a predicate deleted by $\Phi(o_i)$ (and not added back) is the same (in terms of both name and variable symbols) as a predicate in the precondition of $\Psi(o_j)$, then the macro $o_{i,j}$ is unsound.

7.2 Representing macros in ASP

In this section, we describe how macros are encoded in ASP. In order to ease the descriptions, we emphasize the relation between our SAES encoding and common concepts in planning. In particular, atoms appearing in the head of rules r_{16}, r_{18}, and r_{23} represent the *actions*, whereas their bodies represent *preconditions* that must be satisfied in order to select that action. Rule r_{17} represents the *effect* of the action described in r_{16}; rules r_{19}–r_{22} and r_{32} represent the effects of the action in rule r_{18}; and rules r_{24}–r_{25}, r_{31}, and r_{32} represent the effects of the action in rule r_{23}. *Fluents* are all the atoms appearing in the encoding that contains the variable representing the time, in our case usually referred to as `T`.

In ASP, a macro is encoded by a single *choice* rule, which implicitly represents multiple actions, and by several normal rules needed to model their effects. The *head* of the choice rule contains a single fresh atom, whose variables are all the one appearing in the body of the choice rule, whereas its *body* is built as described in the following paragraph. Given a rule r_i representing an action, $pre(r_i)$ denotes the rule body. Intuitively, it represents the conditions that must hold in order to activate the action represented by the rule. Moreover, $del(r_i)$ (respectively, $add(r_i)$) represent all the atoms which are set as false (respectively, true) whenever the conditions denoted by $pre(r_i)$ hold. Therefore, a macro

$r_{i,j}$ is constructed by assembling the rules representing single actions and by generating $pre(r_{i,j})$, $del(r_{i,j})$, and $add(r_{i,j})$, as follows:

- $pre(r_{i,j}) = pre(r_i) \cup (pre(r_j) \setminus add(r_i))$,
- $del(r_{i,j}) = (del(r_i) \setminus add(r_j)) \cup del(r_j)$,
- $add(r_{i,j}) = (add(r_i) \setminus del(r_j)) \cup add(r_j)$.

where r_i and r_j are two distinct rules. As a consequence, for a macro $r_{i,j}$, the body of the choice rule is represented by $pre(r_{i,j})$.

Example 1

As an example, let us consider the two rules r_{16} and r_{18}, which correspond to the two actions `move_link_to_central` and `take_links_to_move`. The aim is to create the macro `linkToCentral_take` combining such two actions. Therefore, $pre(r_{16})$ and $pre(r_{18})$ correspond to the bodies of rules r_{16} and r_{18}, that is, they are defined as follows:

$$pre(r_{16}) = \{\texttt{link(L1)}, \texttt{joint(J1)}, \texttt{gripper(G2)}, \texttt{time(T)}, \texttt{free(G2,T)},$$
$$\texttt{connected(J1,L1)}, \texttt{not in\_hand(L1,T)}, \texttt{not in\_centre(J1,T)}\}$$
$$pre(r_{18}) = \{\texttt{link(L1)}, \texttt{link(L2)}, \texttt{joint(J1)}, \texttt{gripper(G1)}, \texttt{gripper(G2)},$$
$$\texttt{in\_centre(J1,T)}, \texttt{free(G1,T)}, \texttt{free(G2,T)}, \texttt{connected(J1,L2)},$$
$$\texttt{not in\_hand(L1,T)}, \texttt{not in\_hand(L2,T)}, \texttt{connected(J1,L1)},$$
$$\texttt{L1<>L2}, \texttt{G1<>G2}\}.$$

Afterward, $del(r_{16})$ and $del(r_{18})$ contain all the atoms which are set to false after the corresponding actions have been selected. In our example, no atoms are set to false because action `move_link_to_central` is selected, therefore $del(r_{16}) = \emptyset$. Instead, `free(G,T)` is set to false because `take_links_to_move` is selected from rule r_{32}, that is, since only one action can be selected at each time step. Therefore, $del(r_{18}) = \{\texttt{free(G,T)}\}$. Finally, $add(r_{16})$ and $add(r_{18})$ correspond to atoms which are set to true after the corresponding actions have been selected. In our example, action `move_link_to_central` leads to atom `in_centre(J,T)` holding true (from rule r_{17}), whereas `take_links_to_move` leads to atom `in_hand(L1,T)`, `in_hand(L2,T)` (from rules r_{19} and r_{20}), `grasped(G1,L1,T)` and `grasped(G2,L2,T)` (from rules r_{21} and r_{22}) holding true. Therefore:

$$add(r_{16}) = \{\texttt{in\_centre(J1,T)}\}$$
$$add(r_{18}) = \{\texttt{in\_hand(L1,T)}, \texttt{in\_hand(L2,T)},$$
$$\texttt{grasped(G1,L1,T)}, \texttt{grasped(G2,L2,T)}\}.$$

Now we are ready to define $pre(r_{16,18})$, $add(r_{16,18})$, and $del(r_{16,18})$, which are as follows:

$$pre(r_{16,18}) = \{\texttt{link(L1)}, \texttt{link(L2)}, \texttt{joint(J1)}, \texttt{gripper(G1)}, \texttt{time(T)},$$
$$\texttt{gripper(G2)}, \texttt{not in\_centre(J1,T)}, \texttt{free(G1,T)},$$
$$\texttt{free(G2,T)}, \texttt{not in\_hand(L1,T)}, \texttt{not in\_hand(L2,T)}$$
$$\texttt{connected(J1,L1)}, \texttt{connected(J1,L2)}, \texttt{L1<>L2}, \texttt{G1<>G2}\}$$
$$del(r_{16,18}) = \{\texttt{free(G,T)}\}$$
$$add(r_{16,18}) = \{\texttt{in\_centre(J1,T)}, \texttt{in\_hand(L1,T)}, \texttt{in\_hand(L2,T)},$$
$$\texttt{grasped(G1,L1,T)}, \texttt{grasped(G2,L2,T)}\}.$$

Therefore, the choice rule representing the macro $r_{16,18}$ is as follows:

```
{linkToCentral_take(L1,L2,J1,G1,G2,T)} :-link(L1),link(L2),
    gripper(G1),gripper(G2),time(T),free(G1,T),free(G2,T),
    not in_centre(J1,T),not in_hand(L1,T),not in_hand(L2,T),
    L1<>L2,G1<>G2,joint(J1),connected(J1,L1),connected(J1,L2).
```

m_1 `{linkToCentral_take(L1,L2,J1,G1,G2,T)} :-link(L1),link(L2),joint(J1),`
        `gripper(G1),gripper(G2),time(T),free(G1,T),free(G2,T),`
        `not in_centre(J1,T),not in_hand(L1,T),not in_hand(L2,T),`
        `L1<>L2,G1<>G2,connected(J1,L1),connected(J1,L2)`

m_2 `{changeAngle_release(L1,L2,J1,G1,G2,A1,A2,T)} :-link(L1),link(L2),`
        `joint(J1), gripper(G1), gripper(G2), angles(A1), angles(A2),`
        `in_centre(J1,T), not free(G1,T), not free(G2,T), in_hand(L1,T),`
        `in_hand(L2,T), connected(J1,L1), connected(J1,L2),`
        `grasped(G1,L1,T), hasAngle(L1,A2,T), grasped(G2,L2,T),`
        `L1<>L2, G1<>G2.`

m_3 `{grasp_changeAngle_release(L1,L2,J1,A1,A2,G1,G2,T)}:- link(L1),`
        `link(L2), joint(J1), angles(A1), angles(A2), gripper(G1),`
        `gripper(G2), time(T), in_centre(J1,T), free(G1,T), free(G2,T),`
        `connected(J1,L1), connected(J1,L2), in_centre(J1,T),`
        `hasAngle(L1,A2,T), time(T), L1<>L2, G1<>G2.`

Fig. 11. Macros encoding for the extended scenario.

7.3 *Macros for the extended scenario and their ASP encoding*

While evaluating robot plans related to problem instances in the extended scenario, we noticed that some of the actions could be merged. In fact, we observed typical sequences of actions corresponding to the manipulation of links. Leveraging on this observation, we modified the SAES encoding to include such macros. From now on, we will refer to the new encoding as *Macro Action Extended Scenario* (MAES). This section describes the macros we have defined, and then how they have been encoded in ASP, following the example given in the previous subsection. The complete MAES encoding can be found in online Appendix A.

Macros for the Extended Scenario. The following macros have been considered:

- `linkToCentral_take` is shown in Example 1, and it is the composition of the action `move_link_to_central`, which displaces the articulated object so that the joint in-between the links which must be manipulated can be located in the center of the robot manipulation space, and the action `takes_links_to_move`, which grasps the links to be manipulated. Since links may not be successfully grasped by the robot unless they were in the robot manipulation space, this macro aims at providing a single rule for cases whereby links are not in the right position.
- `changeAngle_release` is the composition of `changeAngle`, which changes the orientation of a link, and `release_links`, which releases the currently grasped link. This macro provides a single rule for cases whereby it is necessary to act on a link and release it afterward.
- `grasp_changeAngle_release` represents the composition of the following actions: `take_links_to_move`, `changeAngle`, and `release_links`. This macro provides a single action for cases when it is necessary to act on a link that was already at the center of robot manipulation space.

Macros Encoding for the Extended Scenario. The MAES encoding is an improved version of the SAES encoding, and it is based on principles similar to those employed in the SAS

and SAES encodings. The general structure and the way the MAES encoding deals with robot manipulation actions are the same shown in Figure 6. Figure 11 partially reports how the three macros introduced above are encoded in ASP, considering only preconditions and choice rules, and not showing the effects. To give an overall idea, *MAES* encoding is composed of these three macros, and their effects plus (modified versions of) the rules from r_{26} to r_{30}, which are necessary to ensure that only one action is selected at each time step, the rules from r_{31} to r_{33}, and eventually r_{15} that ensure the goal is reached. Rules from r_{26} to r_{33} must be modified according to other rules of the encoding, for example, rules from r_{26} to r_{29} are reduced to just three rules, one for each macro. As already stated, the full MAES encoding is reported and explained in online Appendix A.

8 Validation in the extended scenario and experimental analysis

In this section, we present and discuss the results of the analysis we carried out on the extended scenario. We first present an experimental validation of the whole framework in the extended scenario, in a way similar to what we have done in Section 5 for the simple scenario, and then we discuss an analysis about the contribution of macros on the *Action Planning* module.

8.1 Validation in the extended scenario

We employed the same experimental setting already described in Section 5. Figure 12 shows the knowledge base of one sample problem instance, as well as two excerpts of the answer set produced by Clingo when using the SAES and MAES encoding, respectively. From the figure, it is possible to appreciate the differences in the knowledge base with respect to the one ones in the simple scenario.

We can compare the solutions generated by SAES and MAES: in the plan generated by the SAES encoding, a_{s1} denotes the fact that the robot must take links 3 and 4 with the left and the right grippers, respectively; in a_{s2}, the orientation of link 4 must change from 60° to 0°; a_{s3} foresees that the orientation of link 4 must change from 0° to 300°; in a_{s4}, the two links must be released; in a_{s5}, joint 4 (located between links 4 and 5) must be placed at the center of robot manipulation space; according to a_{s6}, the robot must grasp links 4 and 5 with the left and the right grippers, respectively; finally, in a_{s7}, the orientation of link 5 must change from 0° to 300°. Instead, according to the plan generated using the MAES encoding, a_{m1} requires the robot to grasp links 3 and 4 with the left and the right grippers, respectively, that the orientation of link 4 be changed from 60° to 0°, and that links are released; according to a_{m2}, the robot must grasp links 3 and 4 with the left and the right grippers, respectively, change the orientation of link 4 from 0° to 300°, and then release the links; in a_{m3}, the two links must be grasped; in a_{m4}, the orientation of link 5 must be changed from 0° to 300°.

While the two solutions share a number of similarities, there are nonetheless few remarkable differences. It is possible to notice that with the macros it is not possible to act on a link without releasing it just afterward: this leads to a small unnecessary idle time if the robot has to move a link two times in a row since it has to release it and grasp it

```
link(1..5).  joint(1..4).
#const granularity = 60.  time(0..timemax).
angle(0).  angle(60).  angle(120).
angle(180).  angle(240).  angle(300).
connected(1,1).  connected(1,2).  connected(2,2).  connected(2,3).
connected(3,3).  connected(3,4).  connected(4,5).  connected(4,5).
hasAngle(1,0,0).  hasAngle(2,90,0).  hasAngle(3,0,0).
hasAngle(4,60,0).  hasAngle(5,120,0).
goal(1,0).  goal(2,90).  goal(3,0).  goal(4,300).  goal(5,300).
```

$a_{s1}:$ `take_links_to_move(4,3,3,0,1,0)`
$a_{s2}:$ `changeAngle(4,3,3,0,60,0,1,1)`
$a_{s3}:$ `changeAngle(4,3,3,300,0,0,1,2)`
$a_{s4}:$ `release_links(3,4,3,1,0,3)`
$a_{s5}:$ `move_link_to_central(4,4,0,4)`
$a_{s6}:$ `take_links_to_move(5,4,4,1,0,5)`
$a_{s7}:$ `changeAngle(5,4,4,300,0,1,0,6)`

$a_{m1}:$ `grasp_changeAngle_release(4,3,3,0,60,0,1,0)`
$a_{m2}:$ `grasp_changeAngle_release(4,3,3,300,0,0,1,1)`
$a_{m3}:$ `linkToCentral_take(5,4,4,4,0,1,2)`
$a_{m4}:$ `changeAngle_release(5,4,4,0,1,300,0,3)`

Fig. 12. An example of problem instance in the extended scenario: the knowledge base of the problem solved with both SAES and MAES (top); two excerpts of the answer set returned by Clingo, respectively, when SAES (middle) or MAES (bottom) are employed.

again. We avoid that problem in the SAES encoding since the two actions `changeAngle` and `releaseLinks` are not encapsulated in a macro.

Figure 13 illustrates the execution of the solution generated using the MAES encoding (from top-left to bottom-right), on the sample instance at the top of Figure 12, which produces the (partial) answer set at the bottom of the same figure. Starting from the initial configuration of the articulated object (top-left), the up row represents the three steps associated with macro action a_{m1}, that is, grasping, link angle change, and release, respectively. Next, the first three figures in the mid row represent action a_{m2}, which is almost equivalent to action a_{m1}. The last figure in the mid row and first two figures of the bottom raw represent three steps of action a_{m3}, which brings the fourth joint to the center of robot manipulation space, whereas the third figure in the bottom row shows the changing orientation step of action a_{m4}. It is noteworthy that even if moving the object toward the center of the manipulation space strictly requires only one gripper, during action execution we hard-code the use of two grippers to increase precision. Finally, the bottom-right figure shows the final configuration corresponding to the required goal configuration of the five-link articulated object. It is important to notice that, for the sake of brevity, we skipped some of the steps involved in actions a_{m2} and a_{m3}. It is interesting to notice, however, although the steps shown in Figure 13 would have been skipped by a SAES-generated plan (see Figure 12), therefore reducing the robot idle time, that the flow of the actions during the execution process of both MAES and SAES is almost equivalent even if the organization of the plan is different. However, we will see in the next subsection the computational advantages that MAES brings.
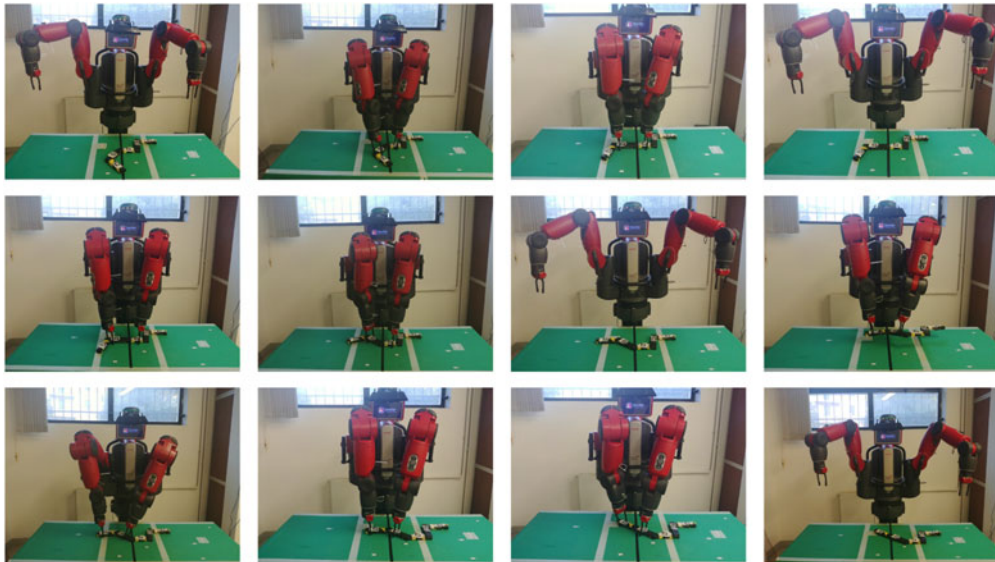
Fig. 13. Execution of a problem instance in the complex scenario.

8.2 Experimental analysis on the Action Planning module

In order to obtain an assessment of the capabilities of the developed encodings, we selected and applied to the new, extended scenario some of the 280 instances generated for the simple scenario. Eventually, we considered 180 problem instances with the number of links varying from 4 to 12, and with orientation granularity values of 4, 6, 8, or 12 possible angles, 5 instances for each pair (number of links, granularity). For each problem instance, a time limit of 300 s and a memory limit of 16 GB was applied. Clingo was used to solve the ASP-encoded instances. All the experiments have been conducted on Intel i7-4790 CPU and Linux OS.

We compared the performance of the considered encodings using *coverage*, that is, the percentage of solved instances, and the Penalised Average Run-time (PAR10) score. The latter is a metric usually exploited in machine learning and algorithm configuration techniques. It trades off coverage and run time for solved problems: if an encoding e allows the solver to solve an instance Π in time $t \leq T$ ($T = 300\,s$ in our case), then $PAR10(e, \Pi) = t$, otherwise $PAR10(e, \Pi) = 10 \times T$ (i.e., 3000s in our case).

Table 1 summarizes the results achieved by Clingo to solve instances encoded in SAS, SAES, and MAES. It is noteworthy to recall that SAS is much more simplistic than the other encodings, as it ignores the position of the links to be manipulated and considers high-level actions which have to be broken down into a large number of low-level elementary actions. On the contrary, SAES and MAES provide a more detailed and rich description of the problem, which allows to generating plans easier to be put in place by the robot. Therefore, a direct comparison between SAS and SAES/MAES is not straightforward, but it is nonetheless interesting to analyze also the results obtained by SAS. Unsurprisingly, the SAS encoding solves a larger number of instances and is generally the fastest among the considered encodings. Due to the above-mentioned issues, the excellent planning performance of SAS comes at the cost of a potentially large number of failures

Table 1. *Results, in terms of PAR10 and coverage, achieved by the considered encodings on the tested instances. Instances are grouped according to the number of links of the articulated object to be manipulated (rows) and the granularity of the angular values, with five instances for each pair (number of links, granularity). Cases not solved by any considered encoding are omitted*

| | Number of allowed orientations: 4 | | | | | |
| | PAR10 | | | Coverage | | |
	SAS	**SAES**	**MAES**	**SAS**	**SAES**	**MAES**
4	0.02	2.0	0.3	100.0	100.0	100.0
5	0.06	622.5	6.6	100.0	80.0	100.0
6	615.54	1852.7	676.3	100.0	40.0	80.0
7	197.31	1817.1	1505.8	80.0	40.0	40.0
8	18.97	2411.9	1254.3	100.0	20.0	60.0
9	92.41	3000.0	3000.0	100.0	–	–
10	1335.49	3000.0	3000.0	60.0	–	–
11	1803.77	3000.0	3000.0	40.0	–	–
12	2401.08	3000.0	3000.0	20.0	–	–

| | Number of allowed orientations: 6 | | | | | |
	SAS	**SAES**	**MAES**	**SAS**	**SAES**	**MAES**
4	0.04	35.6	16.9	100.0	100.0	100.0
5	2.028	1800.1	36.3	100.0	40.0	100.0
6	628.962	3000.0	2400.6	80.0	–	20.0
7	637.682	3000.0	3000.0	80.0	–	–
8	1816.754	3000.0	3000.0	40.0	–	–
9	2718.2	3000.0	3000.0	20.0	–	–

| | Number of allowed orientations: 8 | | | | | |
	SAS	**SAES**	**MAES**	**SAS**	**SAES**	**MAES**
4	6.762	671.7	665.6	100.0	80.0	80.0
5	43.816	3000.0	1860.8	100.0	–	40.0
6	1239.81	3000.0	2454.9	60.0	–	20.0
7	1822.898	3000.0	3000.0	40.0	–	–

| | Number of allowed orientations: 12 | | | | | |
	SAS	**SAES**	**MAES**	**SAS**	**SAES**	**MAES**
4	614.232	3000.0	101.2	80.0	–	80.0
5	1877.986	3000.0	3000.0	40.0	–	–
6	1232.116	3000.0	3000.0	40.0	–	–
7	2809.50	3000.0	3000.0	20.0	–	–

in execution, since the encoding does not take into consideration the effective boundaries of the robot workspace. During the execution of the computed plans, the robot may have to perform actions on joints which are at the limit of the robot manipulation space, therefore causing the robot arms to singularity positions and to stop plan execution.

It is trivial to notice that the more links the articulated object has, the easier it is to experience such failures.

The comparison of the performance achieved by Clingo when using SAES and MAES can shed some light on the usefulness of macros. It can be noticed that macros allow Clingo to solve a larger number of instances, and that macros are generally helpful in improving the run time. This is true regardless of the granularity considered for allowed link orientations and of the number of considered links. As an example, with a granularity value of 12 (i.e., joint's angles can be modified by 30° per movement), the use of macros allow Clingo to solve 80% of instances of size 4; no instances can be solved using SAES, instead.

In summary, it is evident from Table 1 that macros improve considerably planning performance as well as run-time success rate. The introduction of macros leads to better run-time performance, but at the cost of reintroducing idle robot time in certain specific cases. This is aligned with the results achieved in automated planning when exploiting macros: run time and coverage performance of planners tend to improve, but the quality of generated plans may be negatively affected due to repetitions and sub-optimalities (Chrpa and Vallati 2019).

9 Related work

In Sections 4.3 and 6.2, we have shown two of our encodings of the Action Planning module, for the simple and extended scenario, respectively, with one particular manipulation mode and search strategy. In online Appendix A, we provide the encoding exploiting macro actions, with the same manipulation mode and search strategy. However, as we already stated, we have designed a series of encodings, including different search strategies. In particular, after imposing a reasonable `timemax`, we have devised further encodings implementing (*i*) the first strategy based on the algorithm *optsat* (Giunchiglia and Maratea 2006; Di Rosa *et al.* 2008; 2010), whereby the heuristic of the solver is modified so as to prefer plans with increasing length, and (*ii*) a second strategy employing a choice rule to select the time step, of course possibly losing optimality (see also Dimopoulos *et al.* (2017)).

In Capitanelli *et al.* (2017; 2018), a similar framework based on automated reasoning methodologies was presented. The framework employs a Description Logic (DL)-based knowledge representation framework and the Planning Domain Definition Language (PDDL), as well as automated planning engines for the *Action Planning* module, whereas we use an ASP-based uniform language and approach for the whole framework, leaving to the Motion Planner module handling the kinematics problems. Moreover, differently from most of our approaches, models and solvers employed in Capitanelli *et al.* (2017; 2018) are not guaranteed to generate the shortest length plans, which is indeed important, given that in this context action execution is quite time-consuming.

The ASP-based architecture described in this paper can be integrated with ROSo-Clingo (Andres *et al.* 2015), which is a framework combining the ASP solver Clingo (version 4) with the ROS middleware. In particular, it provides a high-level ASP-based interface to control the behavior of a robot and to process the results of actions execution. In our framework, the interaction with ROS is handled by a software adapter developed *ad hoc*. Furthermore, our architecture can be integrated with telingo (Cabalar *et al.* 2019), an extension of the ASP system Clingo with temporal operators over finite

linear time, which automatically uses the multishot interface of Clingo, and therefore it might be useful to further simplify our architecture and improve performance.

ASP has been employed in different domains, including Robotics (Gebser *et al.* 2013; Erdem *et al.* 2013; Andres *et al.* 2015; Erdem *et al.* 2015; Schäpers *et al.* 2018), encompassing various application domain. However, in the current literature, the goal is not the validation and exploitation of the techniques on a real robot, as it is in our case. For a recent overview, the interested reader is referred to Erdem and Patoglu (2018).

Focusing on planning encodings, recently the Plasp system Dimopoulos *et al.* (2017) has been further extended with both SAT-inspired and genuine encodings. Some of them have helped reduce the (still existing) gap with automated planning techniques. Our aim in the design of the encoding for our scenario is to obtain a working solution for the problem at hand, rather than achieving the best-performing solution. Nonetheless, results discussed in Dimopoulos *et al.* (2017) could be employed to further speed up our *Action Planning* module.

An important line of research in Action Planning focuses on increasing the efficiency of the planning process by reformulating the domain knowledge, with the aim of obtaining models which are more amenable for automated reasoners. Significant work has been done in the area of reformulation for improving the performance of domain-independent planners. *Macro-operators* (Korf 1985; Newton *et al.* 2007; Chrpa 2010; McCluskey and Porteous 1997) are one of the best known types of reformulation in classical planning. They encapsulate in a single-planning operator a sequence of *original* operators. From a technical perspective, an instance of a macro is applicable in a state if and only if a corresponding sequence of operator instances is applicable in that state and the result of the application of the macro instance is the same as the result of applying the corresponding sequence of operator instances. The notion of macros can also be exploited by specifically enhanced planning reasoners. This is the case for MacroFF SOL-EP version (Botea *et al.* 2005), which is able to exploit offline extracted and ranked macros, and Marvin (Coles *et al.* 2007), which generates macros online by combining sequences of actions previously used for escaping plateaus. Such systems can efficiently deal with drawbacks of specific planning engines, in this case the FF planner (Hoffmann and Nebel 2001). However, their adaptability for different planning engines might be low. In this work, we aimed at exploiting macros in ASP following the more traditional solver-independent approach, that is, by modifying the encoding, replacing simple actions with macros.

10 Conclusions

In this paper, we presented an ASP-based framework for the automated manipulation of articulated objects in a 2D workspace by a dual-arm robot manipulator. We demonstrated the validity and usefulness of the proposed approach by running real-world experiments with a real robot in two scenarios. In the second extended scenario, we applied to ASP the concept of macros in order to deal with plan generation and execution more efficiently. Our analysis shows the effectiveness of the proposed ASP-based approach, using Clingo as a solver, and the usefulness of employing macros.

We see several directions for future work. First, we are interested in validating the framework on different dual-arm robots, possibly manipulating different articulated objects. Given the nature of the approach, we expect it to generalize with a reasonably limited effort. We also plan to integrate telingo with our framework. Then, we plan to

develop a mixed encoding composed by simple and macro actions. Furthermore, our instances could be an interesting benchmark domain for ASP competitions, for example, Gebser *et al.* (2017; 2020). Finally, we would like to evaluate our encodings also with other solvers, for example, Alviano *et al.* (2019) and, given that the encodings contain a significant number of arithmetic operations, to explore the possibility of employing CASP encodings and solvers, for example, Clingcon (Banbara *et al.* 2017), EZCSP (Balduccini and Lierler 2017), and EZSMT (Shen and Lierler 2018), to deal with numerical constraints.

All the material presented in this paper, including encodings, instances, results, and validations, can be found at `https://tinyurl.com/ycbp798j`.

Supplementary material

To view supplementary material for this article, please visit http://dx.doi.org/ 10.1017/S1471068420000459

References

ALVIANO, M., AMENDOLA, G., DODARO, C., LEONE, N., MARATEA, M. AND RICCA, F. 2019. Evaluation of disjunctive programs in WASP. In *LPNMR*, M. Balduccini, Y. Lierler, and S. Woltran, Eds. Lecture Notes in Computer Science, vol. 11481. Springer, 241–255.

ANDRES, B., RAJARATNAM, D., SABUNCU, O. AND SCHAUB, T. 2015. Integrating ASP into ROS for reasoning in robots. In *Proceedings of the 13th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2015)*, F. Calimeri, G. Ianni and M. Truszczynski, Eds. Lecture Notes in Computer Science, vol. 9345. Springer, 69–82.

BALDUCCINI, M. AND LIERLER, Y. 2017. Constraint answer set solver EZCSP and why integration schemas matter. *Theory and Practice of Logic Programming 17,* 4, 462–515.

BANBARA, M., KAUFMANN, B., OSTROWSKI, M. AND SCHAUB, T. 2017. Clingcon: The next generation. *Theory and Practice of Logic Programming 17,* 4, 408–461.

BARAL, C. 2003. *Knowledge Representation, Reasoning and Declarative Problem Solving.* Cambridge University Press.

BERTOLUCCI, R., CAPITANELLI, A., DODARO, C., LEONE, N., MARATEA, M., MASTROGIO-VANNI, F. AND VALLATI, M. 2019. An ASP-based framework for the manipulation of articulated objects using dual-arm robots. In *Proceedings of the 15th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2019)*, M. Balduccini, Y. Lierler, and S. Woltran, Eds. Lecture Notes in Computer Science, vol. 11481. Springer, 32–44.

BODENHAGEN, L., FUGL, A. R., JORDT, A., WILLATZEN, M., ANDERSEN, K. A., OLSEN, M. M., KOCH, R., PETERSEN, H. G. AND KRÜGER, N. 2014. An adaptable robot vision system performing manipulation actions with flexible objects. *IEEE Transactions on Automation Science and Engineering 11,* 3, 749–765.

BOTEA, A., ENZENBERGER, M., MÜLLER, M. AND SCHAEFFER, J. 2005. Macro-FF: improving AI planning with automatically learned macro-operators. *Journal of Artificial Intelligence Research 24,* 581–621.

BREWKA, G., EITER, T. AND TRUSZCZYNSKI, M. 2011. Answer set programming at a glance. *Communications of the ACM 54,* 12, 92–103.

CABALAR, P., KAMINSKI, R., MORKISCH, P. AND SCHAUB, T. 2019. telingo = ASP + time. In *Proceedings of the 15th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2019)*, M. Balduccini, Y. Lierler, and S. Woltran, Eds. Lecture Notes in Computer Science, vol. 11481. Springer, 256–269.

CALIMERI, F., FABER, W., GEBSER, M., IANNI, G., KAMINSKI, R., KRENNWALLNER, T., LEONE, N., MARATEA, M., RICCA, F. AND SCHAUB, T. 2020. ASP-core-2 input language format. *Theory and Practice of Logic Programming 20,* 2, 294–309.

CAPITANELLI, A., MARATEA, M., MASTROGIOVANNI, F. AND VALLATI, M. 2017. Automated planning techniques for robot manipulation tasks involving articulated objects. In *Proceedings of the 16th International Conference of the Italian Association for Artificial Intelligence (AI*IA 2017)*, F. Esposito, R. Basili, S. Ferilli, and F. A. Lisi, Eds. Lecture Notes in Computer Science, vol. 10640. Springer, 483–497.

CAPITANELLI, A., MARATEA, M., MASTROGIOVANNI, F. AND VALLATI, M. 2018. On the manipulation of articulated objects in human-robot cooperation scenarios. *Robotics and Autonomous Systems 109*, 139–155.

CHRPA, L. 2010. Generation of macro-operators via investigation of action dependencies in plans. *Knowledge Engineering Review 25,* 3, 281–297.

CHRPA, L. AND VALLATI, M. 2019. Improving domain-independent planning via critical section macro-operators. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence (AAAI 2019)*. AAAI Press, 7546–7553.

CHRPA, L., VALLATI, M. AND MCCLUSKEY, T. L. 2015. On the online generation of effective macro-operators. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI 2015)*. AAAI Press, 1544–1550.

COLES, A., FOX, M. AND SMITH, A. 2007. Online identification of useful macro-actions for planning. In *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS 2007)*, M. S. Boddy, M. Fox, and S. Thiébaux, Eds. AAAI, 97–104.

DARVISH, K., WANDERLINGH, F., BRUNO, B., SIMETTI, E., MASTROGIOVANNI, F. AND CASALINO, G. 2018. Flexible human-robot cooperation models for assisted show-floor tasks. *Mechatronics 51*, 97–114.

DI ROSA, E., GIUNCHIGLIA, E. AND MARATEA, M. 2008. A new approach for solving satisfiability problems with qualitative preferences. In *ECAI*, M. Ghallab, C. D. Spyropoulos, N. Fakotakis, and N. M. Avouris, Eds. Frontiers in Artificial Intelligence and Applications, vol. 178. IOS Press, 510–514.

DI ROSA, E., GIUNCHIGLIA, E. AND MARATEA, M. 2010. Solving satisfiability problems with preferences. *Constraints 15,* 4, 485–515.

DIMOPOULOS, Y., GEBSER, M., LÜHNE, P., ROMERO, J. AND SCHAUB, T. 2017. plasp 3: Towards effective ASP planning. In *Proceedings of the 14th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2017)*, M. Balduccini and T. Janhunen, Eds. Lecture Notes in Computer Science, vol. 10377. Springer, 286–300.

ERDEM, E. AND PATOGLU, V. 2018. Applications of ASP in robotics. *Künstliche Intelligenz 32,* 2–3, 143–149.

ERDEM, E., PATOGLU, V. AND SARIBATUR, Z. G. 2015. Integrating hybrid diagnostic reasoning in plan execution monitoring for cognitive factories with multiple robots. In *Proceedings of the IEEE International Conference on Robotics and Automation, (ICRA 2015)*. IEEE, 2007–2013.

ERDEM, E., PATOGLU, V., SARIBATUR, Z. G., SCHÜLLER, P., AND URAS, T. 2013. Finding optimal plans for multiple teams of robots through a mediator: A logic-based approach. *Theory and Practice of Logic Programming 13,* 4–5, 831–846.

GEBSER, M., JOST, H., KAMINSKI, R., OBERMEIER, P., SABUNCU, O., SCHAUB, T. AND SCHNEIDER, M. 2013. Ricochet robots: A transverse ASP benchmark. In *Proceedings of the 12th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR 2013)*, P. Cabalar and T. C. Son, Eds. Lecture Notes in Computer Science, vol. 8148. Springer, 348–360.

GEBSER, M., KAMINSKI, R., KAUFMANN, B., OSTROWSKI, M., SCHAUB, T. AND WANKO, P. 2016. Theory solving made easy with clingo 5. In *Proceedings of the 32rd Technical Communications of the International Conference on Logic Programming (ICLP 2016 TCs)*, M. Carro, A. King, N. Saeedloei, and M. D. Vos, Eds. OASICS, vol. 52. Schloss Dagstuhl, 2:1–2:15.

GEBSER, M., MARATEA, M. AND RICCA, F. 2017. The design of the seventh answer set programming competition. In *LPNMR*, M. Balduccini and T. Janhunen, Eds. Lecture Notes in Computer Science, vol. 10377. Springer, 3–9.

GEBSER, M., MARATEA, M. AND RICCA, F. 2020. The seventh answer set programming competition: Design and results. *Theory and Practice of Logic Programming 20,* 2, 176–204.

GELFOND, M. AND LIFSCHITZ, V. 1988. The stable model semantics for logic programming. In *Proceedings of 5th the International Conference on Logic Programming (ICLP 1988)*. MIT Press, 1070–1080.

GELFOND, M. AND LIFSCHITZ, V. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Computing 9*, 365–385.

GEREVINI, A. E., SAETTI, A. AND VALLATI, M. 2015. Exploiting macro-actions and predicting plan length in planning as satisfiability. *AI Communications 28,* 2, 323–344.

GIUNCHIGLIA, E. AND MARATEA, M. 2006. Solving optimization problems with DLL. In *Proceedings of the 17th European Conference on Artificial Intelligence (ECAI 2006)*. Frontiers in Artificial Intelligence and Applications, vol. 141. IOS Press, 377–381.

HARNAD, S. 1990. The symbol grounding problem. *Physica D 42*, 335–346.

HEYER, C. 2010. Human-robot interaction and future industrial robotics applications. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 4749–4754.

HOFFMANN, J. AND NEBEL, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research 14*, 253–302.

HOWEY, R., LONG, D. AND FOX, M. 2004. VAL: automatic plan validation, continuous effects and mixed initiative planning using PDDL. In *Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2004)*. IEEE Computer Society, 294–301.

KAUTZ, H. A. AND SELMAN, B. 1992. Planning as satisfiability. In *Proceedings of the 10th European Conference on Artificial Intelligence (ECAI 1992)*, B. Neumann, Ed. John Wiley and Sons, 359–363.

KORF, R. 1985. Macro-operators: A weak method for learning. *Artificial Intelligence 26,* 1, 35–77.

KRÜGER, J., LIEN, T. K. AND VERL, A. 2009. Cooperation of human and machines in assembly lines. *CIRP Annals 58,* 2, 628–646.

LIFSCHITZ, V. 2002. Answer set programming and plan generation. *Artificial Intelligence 138,* 1–2, 39–54.

MCCLUSKEY, T. L. AND PORTEOUS, J. M. 1997. Engineering and compiling planning domain models to promote validity and efficiency. *Artificial Intelligence 95,* 1, 1–65.

NAIR, A., CHEN, D., AGRAWAL, P., ISOLA, P., ABBEEL, P., MALIK, J. AND LEVINE, S. 2017. Combining self-supervised learning and imitation for vision-based rope manipulation. In *Proceedings of the 2017 IEEE International Conference on Robotics and Automation (ICRA 2017)*. IEEE, 2146–2153.

NEWTON, M. A. H., LEVINE, J., FOX, M. AND LONG, D. 2007. Learning macro-actions for arbitrary planners and domains. In *Proceedings of the 17th International Conference on Automated Planning and Scheduling (ICAPS 2007)*, M. S. Boddy, M. Fox, and S. Thiébaux, Eds. AAAI, 256–263.

NIEMELÄ, I. 1999. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence 25,* 3–4, 241–273.

SCHÄPERS, B., NIEMUELLER, T., LAKEMEYER, G., GEBSER, M. AND SCHAUB, T. 2018. ASP-Based Time-Bounded Planning for Logistics Robots. In *Proceedings of the 28th International Conference on Automated Planning and Scheduling (ICAPS 2018)*. AAAI Press, 509–517.

SCHULMAN, J., HO, J., LEE, C. AND ABBEEL, P. 2013. Learning from demonstrations through the use of non-rigid registration. In *Proceedings of the 16th International Symposium on Robotics Research (ISRR 2013)*, M. Inaba and P. Corke, Eds. Springer Tracts in Advanced Robotics, vol. 114. Springer, 339–354.

SHEN, D. AND LIERLER, Y. 2018. Smt-based constraint answer set solver EZSMT+ for non-tight programs. In *Proceedings of the 16th International Conference on Principles of Knowledge Representation and Reasoning (KR 2018)*, M. Thielscher, F. Toni, and F. Wolter, Eds. AAAI Press, 67–71.

THOMAS, A., AMATYA, S., MASTROGIOVANNI, F. AND BAGLIETTO, M. 2018. Towards perception-aware task-motion planning. In *Proceedings of the AAAI 2018 Fall Symposium on Reasoning and Learning in Real-World Systems for Long-Term Autonomy*. Arlington, VA, USA.

THOMAS, A., MASTROGIOVANNI, F. AND BAGLIETTO, M. 2019. Task-motion planning for navigation in belief space. In *Proceedings of the 2019 International Symposium on Robotics Research (ISER)*. Hanoi, Vietnam.

WAKAMATSU, H., ARAI, E. AND HIRAI, S. 2006. Knotting/unknotting manipulation of deformable linear objects. *International Journal of Robotic Research 25,* 4, 371–395.

YAMAKAWA, Y., NAMIKI, A. AND ISHIKAWA, M. 2013. Dynamic high-speed knotting of a rope by a manipulator. *International Journal of Advanced Robotic Systems 10*, 1–12.