IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2024, Yokohama, Japan. Cite as RA-L paper.

Long-Horizon Planning and Execution with Functional Object-Oriented Networks

David Paulius*, Alejandro Agostini*, and Dongheui Lee

***Abstract*—Following work on joint object-action representations, *functional object-oriented networks* (FOON) were introduced as a knowledge graph representation for robots. A FOON contains symbolic concepts useful to a robot's understanding of tasks and its environment for *object-level planning*. Prior to this work, little has been done to show how plans acquired from FOON can be executed by a robot, as the concepts in a FOON are too abstract for execution. We thereby introduce the idea of exploiting object-level knowledge as a FOON for task planning and execution. Our approach automatically transforms FOON into PDDL and leverages off-the-shelf planners, action contexts, and robot skills in a hierarchical planning pipeline to generate executable task plans. We demonstrate our entire approach on long-horizon tasks in CoppeliaSim and show how learned action contexts can be extended to never-before-seen scenarios.**

***Index Terms*—Task and Motion Planning, Service Robotics, Manipulation Planning, Learning from Demonstration**

I. Introduction

An ongoing trend in robotics research is the development of robots that can jointly understand human intention and action and execute manipulations for human domains. A key component for such robots is a knowledge representation that allows a robot to understand its actions in a way that mirrors how humans communicate about action [1]. Inspired by the theory of affordance [2] and prior work on joint object-action representation [3], the *functional object-oriented network* (FOON) was introduced as a knowledge graph representation for service robots [4], [5]. FOONs describe object-oriented manipulation actions through its nodes and edges and aims to be a high-level planning abstraction closer to human language and understanding. They can be automatically created from video demonstrations [6], and a set of FOONs can be merged into a single network from which knowledge can be quickly retrieved as plan sequences called task trees [4].

Although task plans extracted from FOON are too abstract for robot execution, FOON can significantly simplify the generation of knowledge representation for task and motion planning (TAMP) instead of handcrafting PDDL [7] (short for *Planning Domain Definition Language*), as done traditionally. Indeed, a FOON is ideal for deriving *object-level* plans that are agnostic to the robot and its environment, as opposed to *task-level* plans, which considers robot and environment constraints [8]. Doing so requires grounding high-level semantic concepts in FOON to the low-level skills and parameters through which a robot interacts with or understands its actions and world [9], [10]. For example, cooking recipes are object-level plans, but they require task-level plans to ground nouns to object instances in the world and verbs to robot skills.

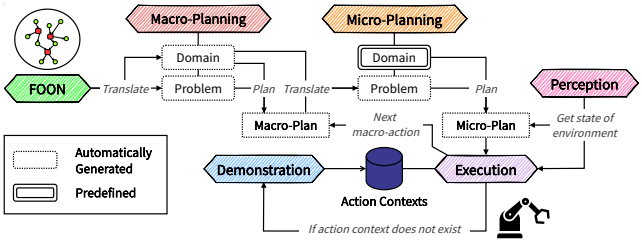

Fig. 1. Overview of our hierarchical planning approach. Domain-independent action sequences in FOON are automatically transformed into PDDL definitions, which are then used for deriving an object-level plan (*macro-plan*) and task-level plan (*micro-plan*). Each step in a macro-plan is grounded to the robot's action space and environment (via perception) as a micro-plan, which is a sequence of robot-executable skills based on action contexts [11].

Therefore, we introduce a two-level hierarchical task planning approach bootstrapped by FOON (Fig. 1). Our approach exploits object-level knowledge to automatically create PDDL planning definitions compatible with off-the-shelf planners and finds a sequence of executable skills, with which a robot can achieve object-level objectives. Further, this approach *deconstructs* a FOON into planning operators, allowing us to generate functional unit sequences beyond those fixed and encoded in a FOON. Our contributions are as follows:

- We introduce an approach to bootstrap task planning by automatically transforming a *high-level*, symbolic FOON into *low-level* planning problems in PDDL.
- We show how our approach finds plans for *novel scenarios*, which may comprise random object configurations or ingredient sets, for the same high-level objective.
- We show how our approach successfully executes long-horizon task plans by leveraging motion dependencies between actions and geometrical consistency via action contexts and an object-centered representation in PDDL.
- We show that our approach has a significantly lower time complexity over classical and HTN planning strategies.

Manuscript received: April 1, 2023; Accepted: June 1, 2023.

This paper was recommended for publication by Editor Aleksandra Faust upon evaluation of the Associate Editor and Reviewers' comments. This research was funded by the Helmholtz Association and the Austrian Science Fund (FWF) Project M2659-N38. David is supported by the Office of Naval Research (ONR) under grant no. N00014-21-1-2584 and Echo Labs. This work began while the authors were members of the Human-centered Assistive Robotics group at the Technical University of Munich, Germany.

David Paulius (Email: dpaulius@cs.brown.edu) is affiliated with the Intelligent Robot Lab at Brown University, Rhode Island, United States.

Alejandro Agostini (Email: alejandro.agostini@uibk.ac.at) is affiliated with the Department of Computer Science, University of Innsbruck, Austria.

Dongheui Lee (Email: dongheui.lee@tuwien.ac.at) is affiliated with the Autonomous Systems group at TU Wien, Austria and also with the Institute of Robotics and Mechatronics, German Aerospace Center (DLR), Germany.

(**Alejandro Agostini and David Paulius are co-first authors.*) (*Corresponding authors: Alejandro Agostini and David Paulius.*)

IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2024, Yokohama, Japan. Cite as RA-L paper.

II. Background

A. Functional Object-Oriented Networks (FOON)

Formally, a FOON is a bipartite graph $\mathcal{G} = \{\mathcal{O}, \mathcal{M}, \mathcal{E}\}$, where $\mathcal{O}$ and $\mathcal{M}$ refer to two types of nodes: *object nodes* and *motion nodes*. Object nodes refer to objects used in tasks, including tools, utensils, ingredients or components, while motion nodes refer to actions that can be performed on said objects. An object node $o \in \mathcal{O}$ is identified by its object type, its states, and, in some cases, its make-up of ingredients or components; a motion node $m \in \mathcal{M}$ is identified by an action type, which can refer to a manipulation (e.g., pouring, cutting, or mixing) or non-manipulation action (e.g., frying or baking). Edges ($e \in \mathcal{E}$) connect these nodes to one another.

Objects may take on new states as a result of executing actions. State transitions are conveyed through edges ($\mathcal{E}$) to form *functional units* (denoted as $\mathcal{FU}$), which describe object nodes before and after an action takes place. Specifically, a functional unit $\mathcal{FU} = \{\mathcal{O}_{in}, \mathcal{O}_{out}, m\}$ contains a set of input nodes $\mathcal{O}_{in}$, a set of output nodes $\mathcal{O}_{out}$, and an intermediary action node m, comparable to the *precondition-action-effect* structure of planning operators (POs) in classical planning [12]. A robot can use a FOON to identify states that determine when an action is completed. Fig. 2 shows two functional units describing a sequence of pouring vodka and ice into a drinking glass. There are notably several object types with multiple node instances, as these object states will change as a result of execution. Each functional unit has the same motion node label of *pour*, yet the objects and effects of each action differ, thus treating them as two separate actions.

FOONs are created by annotating action from observation, such as video demonstrations. We note the objects, actions, and state changes required to achieve a specific goal, such as a recipe, during annotation. This results in a *subgraph*, which is a sequence of functional units (and their respective objects and actions) to fulfill the given goal. Two or more subgraphs can be merged to form a *universal* FOON. Presently, the FOON dataset provides 140 subgraph annotations of recipes with which a universal FOON can be created; these annotations along with helper code are publicly available for use.[1]

B. Task Planning

We adopt the traditional approach for task planning [12] by defining a set of objects (e.g., `cup` or `bowl`) and predicates that encode object properties or relations (e.g., `(on table cup)` – the cup is on the table). Each predicate can be true or false depending on whether these attributes are observed in the scene. The *symbolic* state $\mathcal{S}$ is defined by a set of predicates describing the object configuration in a scenario. Planning operators (POs) describe the changes in the symbolic state via actions and are encoded in the traditional *precondition-action-effect* notation using PDDL [7]. Preconditions comprise the predicates that change by the execution of the PO as well as those that are necessary for these changes to occur. Effects, in turn, describe the changes in the symbolic state after the PO execution as predicates. Fig. 4 provides examples of POs written in PDDL notation. The name of a PO is a *symbolic*

[1]FOON API and Dataset – https://github.com/davidpaulius/foon_api

Fig. 2. Illustration of two functional units for pouring vodka and ice into a glass (best viewed in colour). Each functional unit is discernible by edge style (solid and dashed respectively). Object and motion nodes are denoted by circles and squares respectively. Input-only and output-only nodes are shown in green and purple respectively, while nodes that are both are shown in blue.

action and may contain arguments to ground the predicates to the preconditions and effects. In task planning, a planner uses a description of the *initial state* ($s \in \mathcal{S}$) and a *goal* definition (g) as a set of grounded predicates that should be observed after execution. The planner carries out a heuristic search with these elements by generating causal graphs from the preconditions and effects of POs and yields a sequence of actions called a *plan* that produces changes in s necessary to obtain g.

C. Related Work

There are many notable works that aim to represent knowledge for robots in a way that encourages abstraction for task and motion planning. Frameworks such as KnowRob [13] combine knowledge bases with a query processing engine to allow reasoning over beliefs of the world. Tenorth et al. showed how a robot can use KnowRob to prepare meals, such as pancakes, and form queries over object or action properties. However, they focus on structurally defining this knowledge base and inferring object locations rather than storing or retrieving recipes or task sequences as possible with FOON. Rather, FOON is best used as a schema with reasoning engines or knowledge bases like KnowRob. Ramirez-Amaro et al. [14] investigated how semantic knowledge can be learned from demonstration and then used in planning to imitate demonstrated tasks, such as making pancakes and a sandwich. Although our work does not adopt the same degree of object and activity recognition, knowledge in FOON is agnostic to the robot, and it is only through planning that we obtain a robot-specific task plan suited to its present environment state.

Kaelbling and Lozano-Pérez interleave hierarchical planning with execution using highly abstract behaviours for task planning to accelerate plan generation, but at the expense of planning impasses at execution time [15]. Our approach leverages relevant geometrical constraints at the task planning level to exploit the computational efficiency of planners in generating feasible manipulation plans. Logic programming task planners search for solutions directly in the plan space, rather than in the state space as classic planners, to generate feasible task plans using geometrical constraints [16]. However, these approaches require computationally demanding optimization processes on whole plans using complex dynamic models, making them less suitable for solving long-horizon optimization problems. Other approaches use semantic descriptions of geometrical constraints to evaluate motion feasibility of single actions [17] or sequences of actions [18] that are assessed during task planning using conventional state-based

planners. A task planner finds candidate plans based on these constraints, while a sampling-based motion planner checks action feasibility using geometric reasoning. Instead, we use object-centered predicates to propagate geometrical constraints during task planning in terms of standard relational names that easily map to object poses without the need of external heuristics for geometric reasoning.

Hierarchical task networks (HTN) [12] share many similarities with our approach. HTNs represent abstract tasks called *methods*, comprising a sequence of sub-tasks that are executed in a reactive manner. A planning problem is defined as solving a series of tasks. Such tasks are akin to FOON functional units, as they require a sequence of lower-level actions to accomplish a task. However, methods must be defined for all possible ways of executing tasks, which is not practical in real-world settings. Our FOON-based approach treats each high-level task as its own planning problem, which greatly reduces the planning time complexity without defining fixed-ordered methods.

Previous work explored the encoding of *macro* planning operators into primitive operators for the execution of robotic tasks, combining macro operators and primitives into a single linear planning domain [19] or combining linear planning with reinforcement learning for executing primitives [20]. However, as with HTNs, macro operators are associated with a fixed sequence of primitive operators that are executed in a reactive manner. Manipulation action trees by Yang et al. [21] represent robotic manipulation as a tree for planning and execution. Zhang and Nikolaidis [22] propose executable task graphs, which describe what the robot must do to replicate actions observed from cooking videos, for multi-robot collaboration. However, as their focus was on imitating behaviours from demonstration, they do not show how these graphs could be adapted to novel scenarios as possible with our approach.

III. Task Planning with FOON

Object-level representations like FOON are ideal for bootstrapping TAMP, as they can be applied across robots and domains. Further, functional units integrate well into PDDL notation, as they are encoded using a precondition-action-effect notation that can be directly transformed into PDDL predicates and planning operators. However, we must ground the *domain-independent*, object-level knowledge to how the robot views or interacts with the world as a *domain-specific* representation, where concepts in FOON are grounded to the physical world, relevant object properties, and robot actions.

To use object-level knowledge in FOON for task planning, we devise a two-level hierarchical planning approach (Fig. 1). At the top, *macro*-planning finds a plan skeleton (*macro-plan* – $\mathcal{P}_M$) to prepare a recipe. At the bottom, *micro*-planning finds a sequence of robot skills (*micro-plan* – $\mathcal{P}_\mu$) to execute each FOON action (i.e., macro-plan step) in a given scenario and fulfill the high-level objectives of a macro-plan.

A. Macro-level Planning

1) Overview: The aim of macro-planning is to find a plan schema with which we can perform micro-planning; this is equivalent to finding an *object-level plan* that describes how to use objects to solve a high-level objective. In prior work, we defined a search algorithm that combines breadth-first and depth-first search approaches to directly find a task tree from a universal FOON [4]. However, in this work, we adopt an alternative strategy that transforms FOON into PDDL (example shown in Fig. 3) and then derives solutions from a linear planner to generate a *macro-plan*. We encode each functional unit as planning operators in PDDL, thus *deconstructing* FOON into individual units from which we generate sequences beyond those fixed in a FOON.

(a) Functional unit for pouring vodka

```
(:action pour_vodka_0
 :parameters ()
 :precondition (and
  (under bottle table)
  (on table bottle)
  (in bottle vodka)
  (under vodka bottle)
  (in drinking_glass air)
  (under drinking_glass table)
  (on table drinking_glass) )
 :effect (and
  (in drinking_glass vodka)
  (under vodka drinking_glass)
  (under bottle table)
  (in bottle vodka)
  (under drinking_glass table)
  (on table drinking_glass)
  (not (in drinking_glass air)) ) )
```

(b) Equivalent Macro-Level Definition

Fig. 3. Example of a functional unit for *pouring* vodka into a drinking glass and its equivalent macro-PO in PDDL. Note here that vodka remains in the bottle because the bottle is not emptied as in the functional unit definition.

2) Macro-level Translation: To perform macro-planning, we require domain and problem definitions compatible with linear planners. We do this by *automatically* parsing a FOON: object nodes define object names and predicates, while functional units define macro-planning operators (*macro-PO*). An object node $o \in \mathcal{O}$ is defined by its type and state attributes (e.g., a *drinking glass* (type) is *empty* (attribute) – see Fig. 2). An object o is characterized by one or more predicates of the following two types: 1) *object-centered predicates* [11], [23], which describe geometrical relations or properties for characterizing object configuration spaces for motion planning (e.g., defining what is *in* or *on* objects for manipulation); and 2) *state of matter* predicates, which describe an object's physical state, which is useful for identifying a high-level objective (e.g., an object changes from *whole* to *chopped*).

Object-centered predicates describe poses or locations of objects from each object's perspective as they relate to other objects within the robot's environment, allowing us to consistently represent and propagate geometrical constraints during the heuristic search, thus rendering geometrically feasible plans. These predicates have the form of `(⟨rel⟩ ⟨obj_1⟩ ⟨obj_2⟩)` (see Fig. 3b), where `⟨rel⟩` refers to the spatial relation type, while `⟨obj_1⟩` and `⟨obj_2⟩` refer to the focal object and relative object respectively. We use the spatial relations *in*, *on*, and *under*, as these are typically attributed to object nodes in FOON (see Fig. 3a). For example, we define predicates such as `(on table ⟨obj⟩)` and `(under ⟨obj⟩ table)` to characterize the geometric changes taking place with pick-and-place actions of objects on the table.[2] Additionally, we adopt the convention from prior work [11] to describe an *empty* object as it containing `air` (i.e., `(in`

[2]These object-table relations indicate that the object is `on` the table and that the table is `under` the object to consistently map table-object relations from the effects of a macro-PO to a goal for micro-planning (see Sec. III-B).

⟨obj⟩ air)). By default, if an object node has no *on* state relations, we assume that it is simply present on the working surface (viz., table). In the case where an object *contains* other objects, this is explicitly translated using the in and under relations (e.g., (in bottle vodka) and (under vodka bottle) for a bottle of vodka in Fig. 3b. State of matter predicates characterize the physical properties of objects that are temporally relevant for cooking. For instance, a *whole* object becomes *sliced* as a recipe progresses. Several states in FOON have been identified in related work on state recognition for cooking [24]. Predicates for such states take the form of (⟨som⟩ ⟨obj⟩), where ⟨som⟩ refers to the state of matter type and ⟨obj⟩ refers to the focal object. Examples of these states and their respective predicates are is-whole for the *whole* state, is-sliced for the *sliced* state, and is-mixed for the *mixed* state. The translation from FOON to macro-level PDDL notation is automatic and complete, as spatial relations and physical states are unambiguously mapped to object-centered and state of matter predicates respectively, provided that all states compatible with and related to special skills (e.g., chopping, slicing, mixing) are defined.

3) Finding a macro-plan: We create macro-level domain and problem definitions for planning using the aforementioned predicate types. To construct a macro-domain definition, we transform each functional unit $\mathcal{FU}$ into macro-planning operators by translating the objects in $\{\mathcal{O}_{in}, \mathcal{O}_{out}\} \in \mathcal{FU}$ into precondition and effect predicates. Each macro-PO is assigned a name given by the motion node $m \in \mathcal{FU}$. To construct a macro-problem definition, we define the initial state (s_M) as predicates describing objects initially available for use (i.e., no incoming edges), while we define the macro-goal (g_M) as predicates describing the desired final state from the goal node (e.g., {(in drinking_glass ice), (in drinking_glass vodka)} $\in g_M$ as in Fig. 2). After the domain (macro-POs) and problem (initial state and goal) are defined, we can use a linear planner to find a macro-plan $\mathcal{P}_M$, which contains a sequence of steps (functional units) that should be fulfilled when preparing a recipe. However, a micro-plan is required to execute each action $A \in \mathcal{P}_M$, as each action A has yet to be grounded to a robot's action set. Transforming a FOON into PDDL macro-definitions is achieved without information loss, thus preserving the completeness [12] of the original FOON and guarantees finding identical solutions, granted that all required objects for a recipe are present [4].

B. Micro-level Planning

1) Overview: Once a macro-plan ($\mathcal{P}_M$) has been found, we must perform *micro-planning* to generate a micro-plan ($\mathcal{P}_\mu$) that is executable by a robot. In this process, we treat each functional unit (i.e., $A \in \mathcal{P}_M$) as a micro-planning problem, whose initial state (s_μ) and goal (g_μ) are taken directly from a macro-PO and grounded to objects and states via perception. Along with the problem definition, we must define a micro-domain that details all robot-executable primitives, their necessary preconditions, and their resulting effects as micro-planning operators (*micro-PO* – see Fig. 4). Therefore, with both micro-planning domain and problem definitions, we can acquire a manipulation plan that breaks down each macro-plan action into a sequence of realizable actions.

```
(:action pick
 :parameters (
  ?obj - object
  ?surface - object )
 :precondition (and
  (on ?obj air)
  (under ?obj ?surface)
  (on ?surface ?obj)
  (in hand air) )
 :effect (and
  (on ?obj hand)
  (in ?hand ?obj)
  (under ?obj air)
  (on ?surface air)
  (not (in ?hand air))
  (not (on ?obj air))
  (not (under ?obj ?surface))
  (not (on ?surface ?obj)) ) )
```

(a) Pick

```
(:action place
 :parameters (
  ?obj - object
  ?surface - object )
 :precondition (and
  (on ?obj hand)
  (under ?obj air)
  (on ?surface air)
  (in hand ?obj)
 :effect (and
  (on ?obj air)
  (in ?hand air)
  (under ?obj ?surface)
  (on ?surface ?obj)
  (not (in ?hand ?obj))
  (not (on ?obj hand))
  (not (under ?obj air))
  (not (on ?surface air)) ) )
```

(b) Place

Fig. 4. Examples of *micro-PO* action definitions in PDDL notation defined using object-centered predicates [23]. To account for object sizes (Sec. IV), we defined various *place* POs for small, long, and wide objects.

2) Micro-level Translation: We create a micro-problem corresponding to a macro-PO definition for each step of a macro-plan (i.e., $A \in \mathcal{P}_M$): precondition and effect predicates form the initial state (s_μ) and goal (g_μ) predicates accordingly. As with macro-definitions, we use object-centered predicates and state of matter predicates to characterize the object configuration space for manipulation and physical states that objects undergo as a result of execution, respectively. However, these predicates must be grounded to how the robot perceives or interacts with its world. True and false values of object-centered predicates are directly obtained from object 3D poses and bounding boxes to provide a geometrically complete description of the object configuration space, which allows for a consistent propagation of geometric changes with actions for the generation of feasible plans [11]. Note that the perception mechanisms automatically generate predicates that describe the physical constraints for execution, such as (on table ⟨obj⟩) and (in bowl air). We make some assumptions to guarantee unambiguous grounding and completeness. First, we assume that objects are graspable by the robot from free surfaces. We represent a robot's end-effector with a micro-level object hand, which can be empty or not (i.e., (in hand air) or (in hand ⟨obj⟩)). A robot can pick up an object from its top if no obstacles are on top of it (i.e., (on ⟨obj⟩ air), which transforms to (on ⟨obj⟩ hand) after picking). We also treat the working surface (i.e., table) as a grid of tiles, upon which objects may be present. These cells are present in our experiments (Sec. V), where we use tiles of varying sizes for different objects. Further, macro-level object names are grounded to micro-level object instances, which are observable and usable by the robot (e.g., in the macro-level predicate (in bottle vodka), a bottle object maps to a bottle_vodka instance). These mechanisms guarantee that we find geometrically feasible and complete micro-plans.

A micro-domain contains micro-POs that capture physical preconditions and expected effects of executable skills (e.g., pick, place, pour) in terms of object-centered and state relations. These micro-POs consider constraints like the robot's hand (empty or not empty) and other aspects such as the position and orientation of objects and the available surfaces for robot-object and object-object interactions through the

virtual object `air`. For this work, we manually define micro-level actions; examples of micro-PO are shown in Fig. 4, and further examples can be found in previous work [11].

3) Finding a micro-plan: With a macro-plan, we can acquire a micro-plan, which comprises micro-PO sequences for each macro-PO action, using an off-the-shelf planner such as Fast-Downward [25] alongside micro-level domain and problem definitions. We denote a micro-plan for the i-th macro-action $A_i \in \mathcal{P}_M$ as $\tilde{\mathcal{P}}_\mu(A_i) = \{a_{i_1}, ..., a_{i_m}\}$, where a_i denotes a micro-plan step (skills) and m is the number of skills in the micro-plan necessary to ground A_i in the robot's world. Therefore, we can form a comprehensive micro-plan for the entire task $\mathcal{P}_\mu$ by combining the micro-plans for each macro-action in $\mathcal{P}_M$ as $\mathcal{P}_\mu = \{\tilde{\mathcal{P}}_\mu(A_1), \tilde{\mathcal{P}}_\mu(A_2), ..., \tilde{\mathcal{P}}_\mu(A_n)\}$, where $n = |\mathcal{P}_M|$. It is important to note that a micro-plan for a given macro-PO depends on the configuration of the robot's environment. To put it differently, although micro-POs are pre-defined, the permutation of these actions is solely derived from the linear planner, which distinguishes our hierarchical planning approach from others like HTNs, where methods must be pre-defined in addition to micro-POs (see Sec. V-B).

IV. Execution of a Manipulation Plan

A manipulation plan is made up of a sequence of low-level actions that realizes the effects associated with high-level actions (functional units) in a FOON. These low-level steps are automatically generated using the micro-level problem and domain definition, and they can be linked to motion primitives corresponding to skills. In this work, we associate motion primitives with tuples known as *action contexts* [11] that encode motion dependencies between consecutive actions in a plan for successful execution with an appropriate primitive.

A. Action Contexts

An action context is a data structure that is used to associate a motion trajectory to a sequence of low-level skills. An action context ac is represented as a tuple in the form of $ac = (a_{prev}, a_{now}, a_{next}, \mathcal{T})$, where a_{now} refers to an action being executed, a_{prev} and a_{next} refer to preceding and proceeding actions, and $\mathcal{T}$ corresponds to its associated motion trajectory. Each action (a_{prev}, a_{now}, or a_{next}) is made up of the PO name and its object arguments (as found by the planner), and a set or library of action contexts is denoted as $\mathcal{AC}$. As in prior work [11], trajectories are represented as dynamic movement primitives (DMPs) [26], which use weights as forcing terms to preserve the shape of the original trajectory while allowing different initial and end positions of the robot's gripper.

B. Learning and Executing Action Contexts

When executing a micro-plan with n actions (i.e., $\mathcal{P}_\mu = \{a_1, a_2, ..., a_n\}$) to achieve the effects of a macro-action $A \in \mathcal{P}_M$, a robot can search its library $\mathcal{AC}$ to derive the appropriate primitive for action a_t, given that a robot has executed a prior action a_{t-1} and that it will then execute another action a_{t+1} (if available). To select the appropriate DMP parameters $\mathcal{T}$, we first search for $ac \in \mathcal{AC}$ that matches the present context at a time-step t, where a_{prev} is equal to a_{t-1}, a_{now} is equal to a_t, and a_{next} is equal to a_{t+1}. Action contexts are typically created from grounded actions observed in plan segments [11]. However, encoding action contexts in this manner does not allow reuse in cases where the same motion dependencies are needed for a similar (but not equal) set of objects. Hence, we generalize each ac as a relative coordinate-like tuple, where a_t is treated as the origin point (target), while a_{t-1} and a_{t+1} are treated as points relative to the origin (example shown as Fig. 5). This is inspired by previous work where planning operators were generalized using relative positions to targets in a grid world [27]. In addition, we define a mapping of objects to categories (viz., small, large, or wide) to define similarity across object types. For instance, an ac for a *black pepper shaker* can apply to an object of similar size like a *salt shaker*. With these concepts, we can extend action contexts to novel situations and reuse a suitable set of motion parameters $\mathcal{T}$. However, if no ac in $\mathcal{AC}$ matches the current micro-plan segment, a human demonstration is requested to create a new ac and generate an associated set of DMP parameters [28]. As learning proceeds, the number of demonstrations decreases to zero and the robot eventually becomes fully autonomous [11].

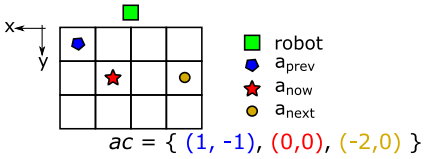

Fig. 5. Illustration of action context generalization. An action context ac can be generalized based on relative positioning of manipulations, where the location of a_{now} is set as the origin $(0, 0)$. The legend (on the right) indicates the symbols used to refer to the target location of each action in ac.

V. Evaluation

To validate our approach, we perform cooking tasks via simulation in CoppeliaSim [29]. For this work, we created a universal FOON made of three subgraphs from the FOON dataset, from which we will perform hierarchical planning to prepare a *Bloody Mary cocktail* and a *Greek salad*. The aim of macro-level planning is to extract a FOON-based plan for each goal (equivalent to a *task tree*), while that of micro-level planning is to find a task-level plan of robot-executable skills tailored to the state of the environment (viz. object locations and configurations). We thus show how this can be applied to randomly generated configurations of the scene while reliably and flexibly using action contexts and motion primitives.

We evaluate our approach with a series of experiments to show that: 1) action contexts can be reused in novel scenarios, 2) FOON-based planning flexibly acquires plans for low-level situations that may not fully match that of the schema proposed by a FOON, and 3) FOON-based planning significantly improves computation time over classical and HTN planning. To address 1) and 2), we measure the average success rate of plan execution for randomized scenes and ingredient subsets, while to address 3), we measure computation time as the overall time taken by the planners to find a solution. An illustration of the universal FOON and demonstration videos for Sec. V-B and V-C are provided as supplementary materials.[3]

[3]Supplementary Materials – https://davidpaulius.github.io/foon-lhpe

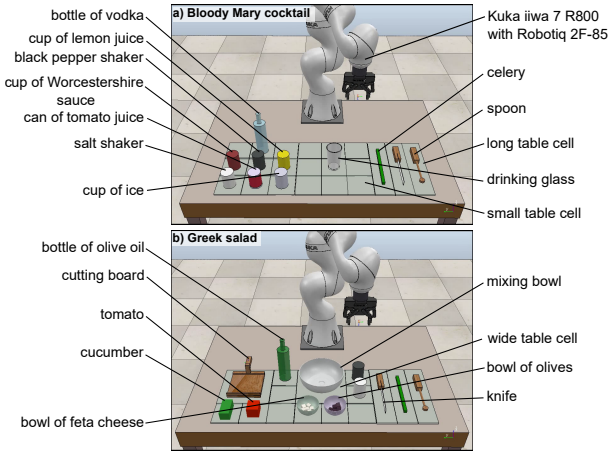

Fig. 6. Layouts for the cocktail and salad tasks in CoppeliaSim.

A. Experimental Setup

Using CoppeliaSim, we designed simple table-top environments with objects and utensils that will be manipulated by a single KUKA LBR iiwa 7 R800 robot arm equipped with a Robotiq 2F-85 gripper. Fig. 6 shows the layout for the cocktail and salad tasks. To make it easier to replicate the recipes while preserving realism, we simplified certain steps in the recipe's FOON for one-armed manipulation; for example, rather than squeezing a lemon for juice, we provide a cup of lemon juice in the scene. This is similar to the cooking principle of *mise en place*. We also fashioned objects, such as the cutting board or knife, for robotic manipulation. We make several assumptions for perception as mentioned in Sec. III. First, objects are placed on cells that discretize the surface; since there are objects of varying sizes (i.e., small, long, and wide), we designed appropriately sized table cells upon which they may be placed. Second, we assume that ingredients in cups or shakers are initialized and symbolically propagated from their starting containers (e.g., *ice* is in the *cup of ice*). We use the following off-the-shelf planners in our experiments: Fast-Downward [25] and PANDA$_{Pi}$[4] (which uses HDDL [30]).

B. Plan Generation for Variable Object Configurations

First, we demonstrate how our approach finds micro-plans for varying object configurations and constraints for the same macro-PO. We perform these experiments on both cocktail and salad scenarios. Fig. 7 shows various scene configurations and micro-plans for `(pour_lemon_juice)` in the cocktail scene (Figs. 7b–d) and `(pick_and_place_tomato)` in the salad scene (Figs. 7f–h) along with their respective micro-plans. These macro-POs are equivalent to the functional units shown as Figs. 7a and 7e (ignoring irrelevant ingredients).

The cases for the *pour* task are as follows: 1) the objects are clear for pouring (Fig. 7b); 2) the drinking glass requires rotation before pouring (Fig. 7c); and 3) the drinking glass requires rotation and the ingredient (cup of lemon juice) is blocked (Fig. 7d). The cases for the *pick-and-place* task are as follows: 1) the cutting board is free of obstacles for placing the tomato on top of it (Fig. 7f); 2) the tomato is obstructed by a salt shaker between it and the cutting board (Fig. 7g); and 3) the cutting board has a stack of obstacles on it that need to be removed prior to placing the tomato (Fig. 7h).

Fig. 7 shows generated plans for different states, yet for the *same* macro-level objective. We provide links to videos for each micro-plan execution in our supplementary materials. When defining this domain as an HTN, we must define methods for every possible way of executing macro-actions. We observed this when implementing an HTN using PANDA$_{Pi}$ and HDDL: just for pouring, we had to account for 8 variations of sub-task sequences, such as when an obstacle was present (blocking a container) or if the target container was not properly oriented. Further, we must define a set of (primitive) actions for an HTN; we use the same set of micro-POs to define these actions. In addition to these actions, we had to define a total of 19 methods for HTN planning. Our approach instead relies solely on a linear planner to find a geometrically feasible micro-plan, whose order is determined by the planner.

C. Transferability to New Scenarios

To demonstrate transferability, we perform two kinds of experiments over 25 trials in variable scenarios: 1) *whole recipe* execution, using all ingredients in the original recipe; and 2) *partial recipe* execution, using random ingredient subsets. Although the *same* object-level plan is found across all trials, each trial results in different manipulation plans due to the configuration of objects (Fig. 6) in the scene (cases such as those in Sec. V-B). In addition, we show that FOON can be flexibly modified at the PDDL level to plan for novel scenarios using fewer objects without creating a new FOON via partial recipe execution. A trial is successful if all objects are manipulated with a suitable action context and motion primitive while avoiding collisions that may cause remaining steps to fail. For example, if the robot knocks a bottle out of the workspace (i.e., table cells) before pouring, then the robot is unable to complete its corresponding macro-PO. Objects stacked on top of others would be placed in a free spot after use to avoid further removing them for remaining steps.

Since objects are randomly configured at the start of each trial, the robot has to rely on learned action contexts. We collected a total of 703 action contexts from demonstration (635 from the cocktail task and an additional 68 from the salad task), which can be generalized using the method from Sec. IV-B. We summarize our results in Table I. In the cocktail task, robot execution was 96% successful for whole execution and 92% for partial execution; in the salad task, robot execution was 80% successful for whole execution and 84% for partial execution. In some trials, the robot failed to complete the task due to object collision, as trajectories encoded by action contexts are not adapted to avoid collisions with objects lying in between manipulated ones. Despite the lack of collision avoidance mechanisms [31], however, the stored shapes were enough to avoid collisions in most cases.

D. Comparison to other Planning Methods

An advantage of using functional units to define PDDL problems is that it simplifies planning, where, rather than composing a single problem definition, our approach transforms

[4]PANDA$_{Pi}$ Planner – https://panda-planner-dev.github.io/

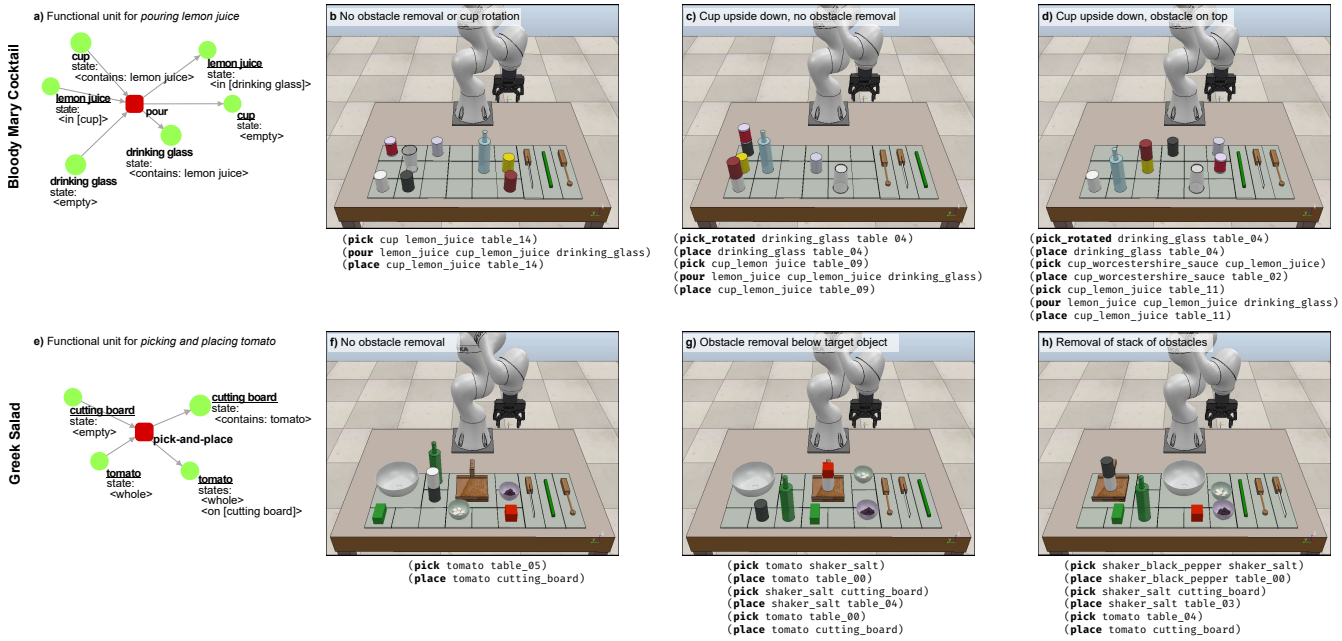

Fig. 7. Qualitative examples of plan variations for cocktail and salad recipes. We highlight various micro-plans for the same macro-level objective (i.e., functional unit). For the cocktail scene, the *macro-PO* is `(pour_lemon_juice)` to pour lemon juice, while for the salad scene, the *macro-PO* is `(pick_and_place_tomato)` for putting a tomato on the cutting board. Videos for each qualitative example are provided in supplementary materials.

TABLE I
SUCCESS RATES WITH RANDOMIZED CONFIGURATIONS OF SCENE OBJECTS FOR WHOLE AND PARTIAL RECIPE EXECUTION

Task	Execution Type	Avg. Plan Length	No. Successful Trials	% Success
Cocktail	Whole	27.9 ± 1.35	$24/25$	96%
	Partial	19.8 ± 3.57	$23/25$	92%
Salad	Whole	34.6 ± 1.78	$20/25$	80%
	Partial	24.9 ± 5.33	$21/25$	84%

each functional unit into smaller problem definitions, which benefits in a significantly reduced time complexity. To support this claim, we compared the average computation time over 10 cocktail scenes for three flavours of planning: (1) FOON-based planning, where we transform each functional unit into macro-problems (our approach in this work); (2) classical planning, where a single problem file is defined with goals of n functional units (where n ranges from 1 to $|\mathcal{P}_M|$); and (3) HTN planning implemented with PANDA$_{Pi}$ and HDDL [30], where we define an HTN with tasks and methods using micro-PO definitions. We use A* search with two heuristics: landmark cut (LMCUT) and Fast Forward (FF)[5]. Running times were measured on a machine running Ubuntu 20.04 with 16 GBs of RAM and an Intel Core i5-8300H processor. A maximum allotted time of 20 minutes was set for each trial. We plot our findings as Fig. 8 using a logarithmic scale to highlight the difference in time complexity between the three approaches.

We can observe that FOON-based planning finds plans in significantly less time than the other methods, as the planner operates with smaller, independent search spaces, which is facilitated by perception for state updates. HTN planning

[5]Details on heuristics – https://www.fast-downward.org/Doc/Evaluator

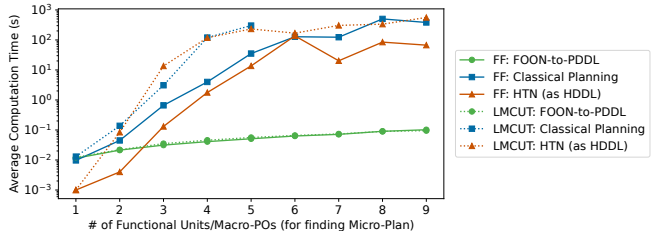

Fig. 8. Graph showing average planning times for solutions found over 10 cocktail scenes for 1) FOON-based planning, 2) classical planning, and 3) HTN planning (HDDL). We compare using landmark cut (LMCUT) and Fast Forward (FF) heuristics for A* search. This graph uses a log-scale to highlight timing differences. Plans beyond 5 functional units were not found within the allotted time using classical planning and LMCUT.

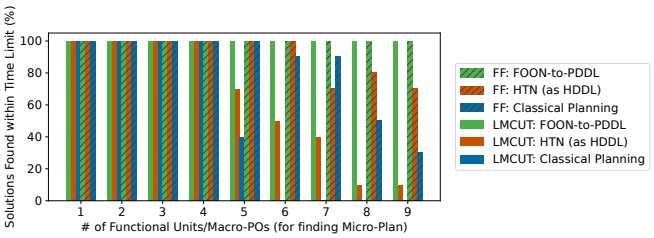

Fig. 9. Graph showing the percentage of scenes for which a solution was found within the allotted time of 20 minutes (best viewed in colour).

found plans in shorter time for smaller problem sizes, but computation times gradually increase for problems larger than 3 functional units. It is also important to note that Fig. 8 shows averages for solutions that were found within 20 minutes; however, many problems of sizes beyond 4 functional units could not be found within the allotted time (see Fig. 9).

Our approach exploits optimal heuristics on smaller problem sets, and thus allows a robot to find and execute a plan in real-time. Further, perception can be used between macro- and micro-actions to monitor the state of the environment. Finally,

FOON schematically enforces a high-level ordering of actions. One key example that requires such ordering is *mixing*. At the macro-level, mixing requires ingredients in a container, but at the micro-level, the only requirement is that the container is free of obstacles on top of it, as it results in the *container* being mixed, i.e., (`is-mixed ⟨cnt⟩`), rather than its contents being mixed. Hence, without a macro-plan, we may acquire a plan where mixing is done before adding all ingredients.

VI. Conclusion

In summary, we introduce an approach to ground rich, object-level knowledge in *functional object-oriented networks* (FOON) for robotic execution of long-horizon tasks in variable scenarios. This approach exploits the efficiency of off-the-shelf planners, action contexts, and object-centered representation in PDDL for the generation of geometrically feasible plans. This is done by a two-step hierarchical decomposition that deconstructs a FOON's functional units into planning operators and predicates in PDDL notation, allowing us to leverage off-the-shelf planners and existing search algorithms. Using object-level representations like FOON to bootstrap task and motion planning allows us to quickly generate flexible solutions that are tailored to the state of the robot's environment.

A. Limitations and Future Work

Despite the exceptional performance of our approach at long-horizon task planning and execution, there are several limitations that we plan to address as future work. One issue is the open-loop nature of the robotic execution, which is unsuited to handle unexpected contingencies inherent to real-robot scenarios when executing micro-plans, such as collisions with objects or external changes to the environment. We will explore re-planning options in the same vein of prior work [27] and include geometric feedback via motion planning in real-world settings for micro-plan execution. Although the DMPs tied to action contexts can reproduce the shape and orientation of trajectories for demonstrated actions, they do not guarantee collision-free executions. We plan to incorporate mechanisms to adapt motion primitives for obstacle avoidance [31].

References

[1] D. Paulius and Y. Sun, "A survey of knowledge representation in service robotics," *Rob. and Aut. Systems*, vol. 118, pp. 13–30, 2019.

[2] J. Gibson, "The theory of affordances," in *Perceiving, Acting and Knowing: Toward an Ecological Psychology*, R. Shaw and J. Bransford, Eds. Hillsdale, NJ: Erlbaum, 1977.

[3] Y. Sun, S. Ren, and Y. Lin, "Object-object interaction affordance learning," *Robotics and Autonomous Systems*, 2013.

[4] D. Paulius, Y. Huang, R. Milton, W. D. Buchanan, J. Sam, and Y. Sun, "Functional Object-Oriented Network for Manipulation Learning," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2016, pp. 2655–2662.

[5] D. Paulius, A. B. Jelodar, and Y. Sun, "Functional Object-Oriented Network: Construction and Expansion," in *2018 IEEE International Conference on Robotics and Automation*, 2018, pp. 5935–5941.

[6] A. B. Jelodar, D. Paulius, and Y. Sun, "Long Activity Video Understanding Using Functional Object-Oriented Network," *IEEE Transactions on Multimedia*, vol. 21, no. 7, pp. 1813–1824, July 2019.

[7] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins, "PDDL – The Planning Domain Definition Language," CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, Tech. Rep., 1998.

[8] D. Paulius, "Object-Level Planning and Abstraction," in *Conference on Robot Learning (CoRL) 2022 Workshop on Learning, Perception, and Abstraction for Long-Horizon Planning*, 2022.

[9] G. Konidaris, "On the necessity of abstraction," *Current Opinion in Behavioral Sciences*, vol. 29, pp. 1–7, 2019.

[10] O. Kroemer, S. Niekum, and G. Konidaris, "A review of robot learning for manipulation: Challenges, representations, and algorithms," *Journal of Machine Learning Research*, vol. 22, no. 30, pp. 1–82, 2021.

[11] A. Agostini, M. Saveriano, D. Lee, and J. Piater, "Manipulation planning using object-centered predicates and hierarchical decomposition of contextual actions," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 5629–5636, 2020.

[12] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning and Acting*. Cambridge University Press, 2016.

[13] M. Tenorth and M. Beetz, "Representations for robot knowledge in the KnowRob framework," *AIJ*, vol. 247, pp. 151–169, 2017.

[14] K. Ramirez-Amaro, M. Beetz, and G. Cheng, "Transferring skills to humanoid robots by extracting semantic representations from observations of human activities," *AIJ*, vol. 247, pp. 95–118, 2017.

[15] L. P. Kaelbling and T. Lozano-Pérez, "Hierarchical Task and Motion Planning in the Now," in *Proc. IEEE International Conference on Robotics and Automation (ICRA)*, 2011, pp. 1470–1477.

[16] M. Toussaint, "Logic-Geometric Programming: An Optimization-Based Approach to Combined Task and Motion Planning," in *International Joint Conference on Artificial Intelligence*, 2015, pp. 1930–1936.

[17] N. T. Dantam, Z. K. Kingston, S. Chaudhuri, and L. E. Kavraki, "An incremental constraint-based framework for task and motion planning," *International Journal of Robotics Research*, vol. 37, no. 10, pp. 1134–1151, 2018.

[18] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "PDDLStream: Integrating Symbolic Planners and Blackbox Samplers via Optimistic Adaptive Planning," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 30, 2020, pp. 440–448.

[19] A. Agostini, E. Celaya, C. Torras, and F. Wörgötter, "Action rule induction from cause-effect pairs learned through robot-teacher interaction," in *International Conference on Cognitive Systems*, 2008, pp. 213–218.

[20] B. Quack, F. Wörgötter, and A. Agostini, "Simultaneously learning at different levels of abstraction," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2015, pp. 4600–4607.

[21] Y. Yang, A. Guha, C. Fermuller, and Y. Aloimonos, "Manipulation Action Tree Bank: A Knowledge Resource for Humanoids," in *2014 IEEE-RAS International Conference on Humanoid Robots (Humanoids)*. IEEE, 2014, pp. 987–992.

[22] H. Zhang and S. Nikolaidis, "Robot Learning and Execution of Collaborative Manipulation Plans from YouTube Cooking Videos," *arXiv preprint arXiv:1911.10686*, 2019.

[23] A. Agostini and D. Lee, "Efficient State Abstraction using Object-centered Predicates for Manipulation Planning," *arXiv preprint arXiv:2007.08251*, 2020.

[24] A. B. Jelodar and Y. Sun, "Joint Object and State Recognition using Language Knowledge," in *2019 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2019, pp. 3352–3356.

[25] M. Helmert, "The Fast Downward Planning System," *Journal of Artificial Intelligence Research*, vol. 26, pp. 191–246, 2006.

[26] A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal, "Dynamical Movement Primitives: Learning Attractor Models for Motor Behaviors," *Neural Computation*, vol. 25-2, pp. 328–373, 2013.

[27] A. Agostini, C. Torras, and F. Wörgötter, "Efficient interactive decision-making framework for robotic applications," *Artificial Intelligence*, vol. 247, pp. 187–212, 2017.

[28] T. Kulvicius, K. Ning, M. Tamosiunaite, and F. Worgötter, "Joining movement sequences: Modified dynamic movement primitives for robotics applications exemplified on handwriting," *IEEE Transactions on Robotics*, vol. 28, no. 1, pp. 145–157, 2011.

[29] E. Rohmer, S. P. N. Singh, and M. Freese, "CoppeliaSim (formerly V-REP): a Versatile and Scalable Robot Simulation Framework," in *Proc. of The International Conference on Intelligent Robots and Systems (IROS)*, 2013, pp. 1321–1326, http://www.coppeliarobotics.com.

[30] D. Höller, G. Behnke, P. Bercher, S. Biundo, H. Fiorino, D. Pellier, and R. Alford, "HDDL: An Extension to PDDL for Expressing Hierarchical Planning Problems," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 06, 2020, pp. 9883–9891.

[31] D. Urbaniak, A. Agostini, and D. Lee, "Combining Task and Motion Planning using Policy Improvement with Path Integrals," in *2020 IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, 2021, pp. 149–155.