A Model for Verifiable Grounding and Execution of Complex Natural Language Instructions

Adrian Boteanu[1], Thomas Howard[2], Jacob Arkin[2] and Hadas Kress-Gazit[1]

***Abstract*— Current methods of grounding natural language instructions do not include reactive or temporal components, making these methods unsuitable for instructions describing tasks as sets of conditional instructions. We introduce the Verifiable Distributed Correspondence Graph (V-DCG) model, which enables the validation of natural language instructions by using Linear Temporal Logic (LTL) specifications together with physical world groundings. We demonstrate the V-DCG model on a physical robot and provide examples of the output our system produces for natural language instructions.**

I. Introduction

Natural language interfaces for robotics are of particular interest in applications designed to allow users to teach new behaviors [1], [2]. Non-expert users programming robots of which they do not have a thorough understanding (e.g. the exact learning algorithm or the perception and manipulation capabilities) face a disconnect between their expectations and the system's ability. In the case of language instructions, the users may not completely state all steps that are part of achieving a goal, considering some to be self-implied.

Within the problem of programming robots through verbal instructions, one key step is assigning physical or symbolic meaning to linguistic utterances, a process known as grounding [3], [4], [5]. Unlike approaches such as entity recognition in text documents [6], which attempts to extract names through text (or other data e.g. relational) analysis without direct knowledge of the physical world, grounding generates a direct mapping between physical entities or actions and utterances. In particular in the case of non-expert users, for the reasons mentioned in the previous paragraph, we argue that a system should holistically evaluate the validity of the grounded instructions before attempting to execute them.

Recent work in grounding language to actions has proposed learning probabilistic graphical models such as the Generalized Grounding Graph (G^3) [7] and the Distributed Correspondence Graph (DCG) [8]. These models learn mappings between parsed language and physical objects from a corpus of examples, each example demonstrating a behavior associated with an instruction. The resulting mappings are superficially representative of the demonstration and the environment, since they use only the current state of the system as input to a generative model. This limits their applicability

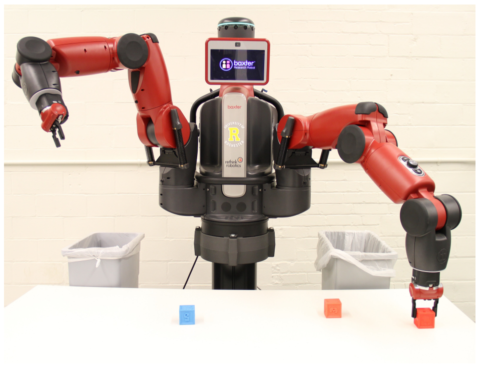


Fig. 1. A Research Baxter robot performing a sorting task using a controller synthesized from natural language using the V-DCG model.

to tasks which contain conditioned execution, i.e. the robot reacting to the environment, such as the following example:

Example: Consider a robot executing a task of sorting cubes of two different colors into their respective bins. At any time during execution the bins can become obstructed by lids, in which case the robot needs to request help from the user to clear the bins. Figure 1 shows the setting of this task. Conveying this task through natural language instructions would require conditional statements. We argue that the robot should be capable of validating the set of instructions as a whole in order to achieve robust behavior that is in conformity with the user's expectations.

To address such scenarios, we introduce the V-DCG grounding model, which extends the DCG model to a new symbolic representation by simultaneously grounding complex instructions to Linear Temporal Logic (LTL) formulae and to physical meanings such as objects and actions. We define *complex instructions* as instruction sets that describe a number of conditionally-dependent requests, which require the robot to asses both the environment's state as well as the current stage of task execution. To illustrate, we include the following two instruction fragments from the corpus we collected (described in Section V):

- "Pick up the blue cube with your right hand [...] Wait until the bin is uncovered, put the blue cube in the right bin and then [...]."
- "[...] Put the boxes in the bins on your right and left. Call for help if the right box is covered. [...]"

The LTL formulae produced by grounding are complemented with *template* formulae describing task and robot constraints, resulting in specifications. Leveraging recent work in verifiable controller synthesis [9], [10], we can

[1]Adrian Boteanu (`ab2633@cornell.edu`) and Hadas Kress-Gazit (`hadaskg@cornell.edu`) are with the Sibley School of Mechanical and Aerospace Engineering, Cornell University, Ithaca, NY 14853. [2]Thomas Howard (`thoward@cs.rochester.edu`) and Jacob Arkin (`j.arkin@rochester.edu`) are with the Hajim School of Engineering and Applied Sciences, University of Rochester, Rochester, NY 14627. A.B and H. K.-G. were supported by NSF NRI award 1427030.

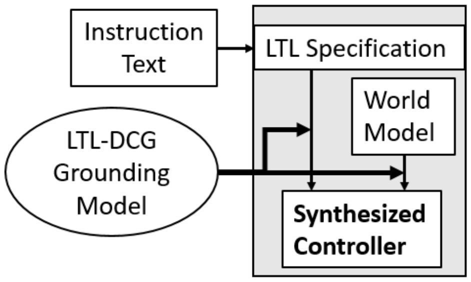


Fig. 2. Block diagram of our approach, showing how a trained V-DCG model is used for grounding novel instructions and synthesizing the grounding output into executable controllers.

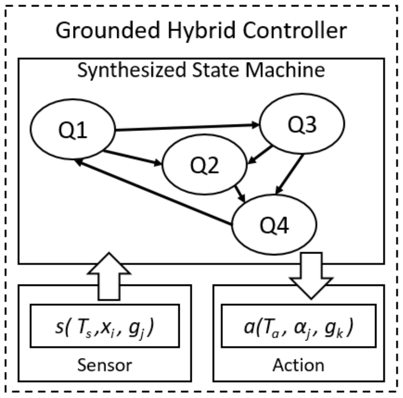


Fig. 3. The V-DCG model grounds natural language instructions to physical environments and LTL specification fragements, which are synthesized into a hybrid controller that can execute the instruction. Sensor and Action functions have types, T_s and T_a, and use to both LTL propositions, x_i and α_i, and to physical groundings, g_j and g_k.

synthesize the specifications into a reactive controller. The resulting controller has a hybrid structure, with the LTL specification synthesized into a state machine that interfaces with low-level functions corresponding to perception (sensors) and actuation (actions) (Figure 3). The low-level functions associated with LTL propositions rely on physical groundings in order to perceive or act on the environment. The outcome of the synthesis step indicates whether the specification is correct or not: if the synthesis step fails, there exists a state in the world that the controller cannot respond to while obeying task assumptions. Such states can be automatically identified [11] and then presented to the user. We note that our method verifies the instruction sequence and not the physical language groundings i.e. if the objects are assigned correct words.

In this work we describe LTL specifications using Structured English in order to allow for simpler text processing from user instructions to specifications [12]. The Structured English specifications are then translated to LTL formulae using an existing framework [10].

We demonstrate the V-DCG model by learning a new model using crowdsourced instructions for a task of sorting cubes by color. The data collection process was done using simulated videos (Figure 7) and the resulting model was used to interpret commands on a physical robot, which can be seen in Figure 1 and in the supplemental video. We note that the V-DCG model maintains the performance of the underlying DCG model in inferring spatial constraints, a feature necessary for distinguishing the target bin for sorting (i.e. to the left or right of the robot).

The remainder of the paper is structured as follows: First, we review relevant literature in Section II and necessary notions on reactive controller synthesis (Section III). Then we describe the V-DCG model and the process of training a new model using natural language instructions (Section IV). We then show how the trained model in conjunction with task templates is used for grounding new instructions (Section IV-B). To complete the hybrid controller, we implemented proposition functions for the perception and actuation primitives corresponding to propositions (Section IV-C). We evaluate our method using a corpus of crowdsourced instructions which we collected (Section V), and conclude the paper with a discussion on the design of our system and future work.

II. Related Work

The V-DCG model extends the existing DCG model to include a formal verification step in addition to grounding language to actions in physical environments. The DCG model was proposed as an improvement to the Generalized Grounding Graph, G^3 [7]. The G^3 model searches a space of all possible assignments of objects and trajectories, thus the search increases in complexity at an exponential rate. The search space of the G^3 model increases with the number of objects, number of possible collections of objects (e.g. "pair of shoes"), as well as the complexity of the robot model.

The DCG model improves the efficiency of this search by inferring constraints that delimit the search space into qualitative areas. Each constraint divides the workspace according to some criteria, for example a circular region around an object corresponds to the phrase "near the box." The intersection of all these constraints is then used to search for a solution using a motion planner. More recent additions to the DCG model exploit the hierarchical structure of human environments, limiting the grounding search space based on the frame of reference to which an utterance is related [13]. Other work in learning groundings of language to robot actions proposes training probabilistic parsers in order to enable more robust instruction parsing [14], [15].

Natural language requests have also been grounded through interactive dialog, in which the robot initiated dialog with users in order to disambiguate requests [1]. A similar approach was shown in conjunction with a pre-trained semantic parser that identified combinatorial categorical grammar (CCG) mappings [16], enabling logical inference through composition. One limitation of these approaches is that they do not learn from demonstrations and require direct interaction for grounding. Such interactive solutions also impose a practical limitation on the language complexity and a relatively flat sentence structure, since each grounding needs to be disambiguated directly by the user.

Our method uses pre-defined task templates in order to complete the LTL formulae to which instructions are grounded. There exists previous work which uses first order logic templates to state high level behaviors for robots manipulation planning [17]. LTL templates have also been used to specify open-world missions, in which behaviors are specified on abstract types instead of instances [18]. Unlike V-DCG, these models do not provide groundings for language, focusing on task specification instead.

III. Preliminaries

We use LTL to validate that grounded instruction sequences are consistent. LTL is a formalism consisting of propositions and a number of operators. Propositions are atomic logical variables that can take a binary value ($true$ or $false$). The operators can be Boolean ($\neg, \wedge, \vee$) and temporal (next, $\bigcirc$, until, U, together with the derived symbols eventually, $\diamond$ and always, $\square$). To specify tasks, we use LTL formulae which are constructed from atomic propositions, $\pi \in AP = X \cup Y$. A formula ϕ is evaluated over an infinite sequence of possible value assignments of its constituent atomic propositions. We use the GR(1) fragment of LTL, which allows for tractable verification [19].

We will now summarize the method we use for reactive controller synthesis, which is presented in detail in [20]. The robot can perform actions abstracted as a set of discrete binary propositions $A = \{a_1, a_2, ...a_n\}$. The action propositions have corresponding action functions, which implement the desired functionality. Whenever an action proposition is active (has a logical value of $true$), the corresponding function commands the robot to perform the action. The robot uses sensor propositions as an abstraction of its perception systems. Similarly to action propositions, the sensor set $S = \{s_1, s_2, ...s_m\}$ is a set of binary logical propositions which are $true$ or $false$ depending on some perception function. In V-DCG, as we detail in Section IV-C, proposition functions further use physical groundings for actuation and perception. In addition, the system contains *memory* propositions which are used to express the internal state of the robot. Unlike previous work which was centered on navigation tasks, the V-DCG model does not use pre-defined "region" propositions to compartment the work space. Instead, we use object locations as motion planning targets, which is accomplished by including object identifiers in the action and sensor proposition functions.

IV. The V-DCG Grounding Model

The process of constructing a V-DCG model for a given task consists of four stages: the first is to define a symbol space which is representative of the task (Section IV-A), followed by parsing and annotating a training set of natural language instructions with groundings and LTL specifications (Section IV-B). Together with this annotation, features for training a log-linear model (LLM) need to be defined. In addition to lexical and spatial features present in the original DCG model, we define features specific to the symbol hierarchy used by V-DCG. These latter features, which the LLM learns when they are positively expressed in the training data, are designed to learn mappings within the symbol hierarchy (Figure 5), as follows: (1) sensor and action types associated with physical object types; (2) sensors and action types associated with a scope, including a LTL formula relating the sensor to the action.

With the symbolic space and training data defined, the remaining steps are to define LTL templates to complete the grounding LTL formulae into specifications, and to implement functions corresponding to the abstract LTL propositions used in the specification (Section IV-C).

A. Symbol Definition

The V-DCG model uses a grounding symbol hierarchy that jointly represents physical and logical symbols (Figure 4). This symbol hierarchy is used to annotate the parse tree corresponding to a sentence. First, we will describe the symbol hierarchy and in Section IV-B we will show how it is used for annotation.

The root of the hierarchy, the *Grounding Set*, is the collection of groundings corresponding to a full instruction. This set is partitioned into *Scopes*, which are used to indicate correspondence between *Sensor* and *Action* symbols, as well as the LTL formulae in which the sensor and action symbols are used. Sensor and action propositions determine the transitions of the state machine synthesized from LTL. The distinction between sensors and actions is that sensors represent variables activated through the robot's perception, while action propositions are activated in order to command high level behaviors.

A sensor specifies a binary proposition and it is associated to one or more *Objects*. We use the notation $s(x_i, g_j)$ to denote that the symbol s is linking the LTL sensor proposition x_i to the physical object grounding g_j by using a function specified by the sensor type. Similarly, an action specifies an LTL proposition and its corresponding objects, if any objects are required. The notation $a(\alpha_i, g_k)$ implies that the LTL action proposition α_i is linked to the physical object grounding g_k through a function that performs instructions specified by the action type.

We defined the following types of *Sensor* symbols for the sorting task:

- observed(x_i, g_i), which is active if the object g_i is on the table and within reach;
- clear(x_i, g_i), which is active if the object g_i is clear of obstructions.

We also defined the following *Action* symbols:

- pick(α_i, g_i), instructing the robot to pickup object g_i;
- drop(α_i, g_i), which commands the robot to perform a drop action above object g_i;
- intervention, which is a pre-define action sequence through which the robot signals it needs assistance.

In addition to sensor and action propositions, the LTL specification also contains *memory* propositions. While the

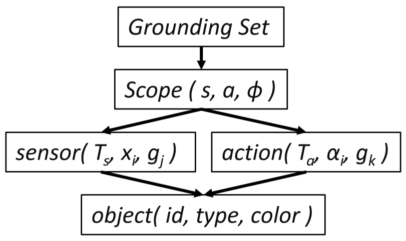

Grounding Set
Scope (s, a, ϕ)
sensor(T_s, x_i, g_j)
action(T_a, α_i, g_k)
object(id, type, color)

Fig. 4. The V-DCG grounding symbol hierarchy. A scope corresponds to a sensor symbol s, an action symbol a and a LTL formula Φ. Both sensors and actions use objects defined by a set of properties (type and color, in our case) and an identifier.

LTL templates include memory propositions, our training model does not include separate symbols for these propositions since they are specific to the domain, do not vary with the user's instructions, and have no physical grounding.

Finally, *Object* symbols describe physical entities in the environment, represented using a collection of discrete properties grouped by type. Within each property type, N/A are special values indicating that the property is unspecified in the annotation, for example the color of the robot's gripper. For the sorting task, we represent objects using a combination of the following:

- Object types = { cube, bin, lid, table, robot torso, left gripper, right gripper, N/A };
- Object colors = { red, blue, N/A }.

B. Instruction Parsing and Annotation

The V-DCG model uses the symbol hierarchy described above in conjunction with a world model. We represent world models as sets of objects defined a pose, a set of symbolic properties (in our case, type and color), and an object id that uniquely identifies an object in the world. The world model is necessary for the system to learn the correspondence between a sensor or action type and the characteristics of the object, and to identify objects that have those characteristics. For example, the red_cube sensor function activates for any object of the type $cube$ with color red, and returns the corresponding object identifiers. Actions that are triggered by the state machine when the red_cube sensor is active will use this object identifier to command the robot to grasp the object.

This symbol representation allows us to state the LTL specification more generally, such that the grounding LTL formulae operate on object types and not instances. Thus, the specification can be used with any objects that fit the properties required by the V-DCG grounding model, and allows for the world model to be expanded at runtime with new objects. A design which we initially considered was to use separate sensor and action propositions for each physical object in the environment instead of operating on object properties; since it imposed a closed world model through the requirement of resynthesizing the specification in order to introduce new objects, we abandoned that design.

```
<instruction text="take the blue cube" />
<VP>
<grounding> <grounding_set>
  <scope ltl_line="do pickup_right if and only if \
                     you are sensing blue and \
                             you are not activating left_gripper">
        <sensor type="observed" invert="false">
          <object type="cube" color="blue" />
        </sensor>
        <action type="pickup" invert="false">
          <object type="robot_right_hand" color="NA"/>
          <object type="cube" color="blue" />
        </action>
</scope> </grounding> </grounding_set>
<VB text="take" />
<NP>
  <grounding> <grounding_set>
        <object type="cube" color="blue" />
  </grounding> </grounding_set>
  <DT text="the" />
  <JJ text="blue" />
  <NN text="cube" />
</NP></VP>
```

Fig. 5. Example of instruction and corresponding annotation in XML format; the annotation was simplified for clarity. The instruction text is contained in an *instruction* element, followed by the parse tree interleaved with the grounding annotation.

We will now show how an instruction is annotated with grounding symbols. This is the format that we used to train a V-DCG model in our evaluation as well as the output of the trained model for new instructions. Figure 5 shows the parse tree and annotation tags corresponding to the instruction *"Take the blue cube."* In order to train new models, we first generated a *grounding space* containing all possible representations that our symbol set can express at all levels in the symbol hierarchy, except for the root (i.e. *grounding set*). This search space is the power set of all possible symbols, of which only a subset are encountered in the training set and thus learned.

When presented with a new natural language instruction, the V-DCG model can produce a corresponding grounding set, which includes LTL formulae associated with each scope. These represent the variable part of the task logic, which changes depending on the user's instruction. To complete the specification, we define templates that contain LTL formulae specific to the domain and the robot. For the sorting task, these constrain the robot to only pickup one object at a time with either gripper. Figure 6 shows the output specification after combining the LTL grounding result with the template LTL formuals. In the figure, we highlighted the lines introduced from the template, the rest of the specification being produced using language grounding. As required by the synthesis framework, the specification is completed by declarations of all propositions that are used.

C. Proposition Functions

As shown in Figure 3, the LTL specification is synthesized into a hybrid controller, which is formed by a high-level state machine and primitive functions corresponding to sensor and action propositions. Working with a discrete time representation, the state machine polls the sensor function at each time step for their value. In return, each sensor function scans the environment for specific conditions and returns a boolean value. Depending on the current state,

```
Actions:
drop_right, 1
pickup_right, 1
intervention, 1

Sensors:
right_bin_clear, 1
blue, 1

Customs:
right_gripper

Spec: # Specification in structured English
robot starts with false
right_gripper is set on pickup_right and
                 reset on drop_right

do pickup_right if and only if you are sensing blue
            and you are not activating right_gripper

do drop_right if and only if you activated right_gripper
            and you are sensing right_bin_clear

if you activated right_gripper and you are not sensing
            right_bin_clear then do intervention
```

Fig. 6. Instantiated template for the annotated instruction shown in Figure 5 and corresponding drop and intervention instruction (not shown for brevity); synthesis settings are excluded as well. The specification starts by declaring all propositions (actions, memory and sensor), followed by the specification logic. The highlighted lines are the domain-specific part of the template.

sensor and action values, the state machine then transitions. In our implementation, separate functions correspond to each sensor and action type (see Section IV-A for a summary of these types).

Sensor functions use the pose and geometry of each object perceived by the robot, together with the object identifier groundings. For example, the function corresponding to the proposition $sensor\_blue$ activates for any object id to which a phrase such as "blue cube" or "blue box" was grounded. We implemented the following sensor functions:

- *observed_function(object_ids)*: tests if either of the argument identifiers is at a minimum height above the ground and within the robot's working space;
- *clear_functions(object_id)*: tests if a destination bin is clear of obstructions, tests if there is any object nearby closer than a threshold, except for the bodies corresponding to the robot and the cubes.

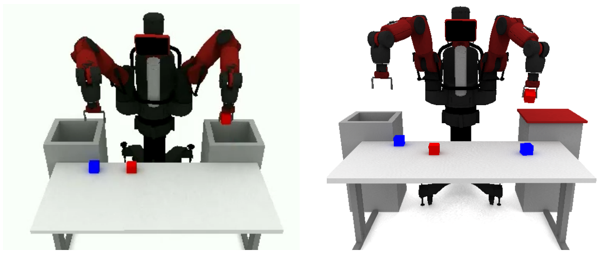
Action functions allow for multiple sub-primitives to be chained into a single function, increasing the modularity of the proposition function system. For example, both grasping and dropping actions include a step which resets the arm to a starting pose. We implemented the following:

- *pickup_function(object_id)*;
- *drop_function(object_id)*;
- *intervention()*.

Both the pickup and drop functions use the target object geometry to compute grasps and drop-off points as targets for the motion planner. The motion sequence for requesting interventions, which can be seen in the supplementary video, consisted of the robot slightly raising and lowering its grippers from the neutral arm position.

V. Evaluation

We will first describe the process through which we collected an instruction corpus for the sorting task. First, we created eight variations of the task environment by changing the number and placement of the cubes. We then executed the task using a hand-coded specification for which we manually provided groundings for each environment variation. We introduced further variance during execution by varying the timing of when the bins would be covered or clear. We recorded videos of the robot executing each task until completion, and then crowdsourced[1] using surveys a total of 50 responses. The surveys instructed participants to watch a video of the robot performing the task and type, using unrestricted language, instructions that they would give to the robot in order to result in the shown behavior.



(a) The task we used in our evaluation involved sorting items into bins. (b) Bins can be obstructed during execution, in which case the robot requests an intervention.

Fig. 7. The simulation environment.

We parsed these responses into 108 sentences, which have a broad range of grammatical complexity and instruction detail, ranging from accurately describing a single step of the task (e.g. *"Pick up the blue cube with your right hand"*) to the sorting groups of cubes (e.g. *"Relocate the center blocks first into the box"*) and expressing symmetry (e.g. *[...] then do the same with the remaining block*). Thus, for demonstrating our system, we selected an initial set of six instructions in order to allow for grammatical commonality between the training samples. The selected sentences had sufficiently similar grammatical structure to enable probabilistic learning. We anticipate that a larger data collection will allow further grammatical patterns to emerge.

We then manually annotated the instructions with groundings corresponding to the symbol hierarchy; an annotation example is shown in Figure 5, simplified for clarity. In total, the annotations consist of 137 object groundings and 25 scope symbols, each scope expressing sensor and action symbols, and LTL formulae.

We trained a V-DCG log-linear model on these instructions. We will now describe the grounding results our model produces:

1) The instruction *'drop the blue cube in the right bin"* is grounded to the LTL formula
   `do drop_right if and only if you activated right_gripper and you are sensing right_bin_clear`
   The resulting specification can be synthesized because it is self-consistent, but because there is no corresponding *pickup*, the instruction is insufficient to accomplish the task. The system alerts the user that the *pickup_right* proposition is not used.

[1]Using Amazon Mechanical Turk (AMT) http://www.mturk.com

2) The instruction *"pick up the remaining red cube and drop in the left bin"* implies both pickup and drop actions:

```
do pickup_right if and only if you are sensing blue
and you are not activating right_gripper
do drop_right if and only if you activated
right_gripper and you are sensing right_bin_clear
```

However, the system outputs a warning indicating that no instruction is referring to red cubes;

3) The instruction *"Pick up the blue block and drop it in the box on your right. Pick up the red block and drop it in the box on your left"* spans two sentences, which are grounded separately. The resulting LTL formulae are combined in a single specification, which can be synthesized. However, a warning related to the unused intervention proposition is issued. Grounding further instructions in which interventions are requested, such as the fragment shown in the Introduction, completes the specification.

Since the V-DCG model is based on the DCG model, it infers spatial object locations. Thus, the location of the target bin relative to the robot is inferred from the world models and does not require to be explicit in the specification. This allows the framework to infer object positions without requiring the instructions to state them, as in the second example. We recorded the attached video by executing a controller synthesized as a result of grounding the third example above (Figure 1).

VI. Conclusion

In this paper we presented the V-DCG model, which expands on the existing DCG grounding framework by introducing a formal synthesis stage which allows for complex instruction sequences to be verified as a whole. The synthesis process, if successful, produces a hybrid controller that is guaranteed to accomplish the task within domain assumptions. Domain assumptions are provided as task templates, which are combined with LTL formulae resulting from the grounding process in order to complete the specification prior to synthesis. We demonstrated our approach by learning groundings for a binary sorting task using crowdsourced natural language instructions. In future work, we plan on conducting a user study to further evaluate the model.

We will now discuss two limitations of our method. First, the V-DCG model uses LTL grounding symbols that are meaningful for a given task. We plan to expand this representation to allow action hierarchies in a manner inspired by Hierarchical Task Networks [21], thus enabling transfer learning between task domains. Second, the verification stage is dependent only on the instructions stated by the user, and cannot determine whether the instructions progress towards the user's intended goal, as indicated in the first example from our evaluation. We plan to use the V-DCG model in conjunction with goal-state learning [22], [23] to detect whether the inferred groundings and specifications also correspond to plausible task goals.

References

[1] S. Lauriar, G. Bugmann, T. Kyriacou, J. Bos, and E. Klein, "Training personal robots using natural language instruction," *IEEE Intelligent systems*, no. 5, pp. 38–45, 2001.

[2] S. Lauriar, G. Bugmann, T. Kyriacou, and E. Klein, "Mobile robot programming using natural language," *Robotics and Autonomous Systems*, vol. 38, no. 3, pp. 171–181, 2002.

[3] S. Harnad, "The symbol grounding problem," *Physica D: Nonlinear Phenomena*, vol. 42, no. 1, pp. 335–346, 1990.

[4] D. Roy, "Semiotic schemas: A framework for grounding language in action and perception," *Artificial Intelligence*, vol. 167, no. 1, pp. 170–205, 2005.

[5] K.-y. Hsiao, S. Tellex, S. Vosoughi, R. Kubat, and D. Roy, "Object schemas for grounding language in a responsive robot," *Connection Science*, vol. 20, no. 4, pp. 253–276, 2008.

[6] P. Singla and P. Domingos, "Entity resolution with markov logic," in *Data Mining, 2006. ICDM'06. Sixth International Conference on*. IEEE, 2006, pp. 572–582.

[7] S. Tellex, T. Kollar, S. Dickerson, M. R. Walter, A. G. Banerjee, S. J. Teller, and N. Roy, "Understanding natural language commands for robotic navigation and mobile manipulation." in *AAAI*, 2011.

[8] T. M. Howard, S. Tellex, and N. Roy, "A natural language planner interface for mobile manipulators," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, 2014, pp. 6652–6659.

[9] S. Chinchali, S. C. Livingston, U. Topcu, J. W. Burdick, and R. M. Murray, "Towards formal synthesis of reactive controllers for dexterous robotic manipulation," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012, pp. 5183–5189.

[10] C. Lignos, V. Raman, C. Finucane, M. Marcus, and H. Kress-Gazit, "Provably correct reactive control from natural language," *Autonomous Robots*, vol. 38, no. 1, pp. 89–105, 2015.

[11] V. Raman, C. Lignos, C. Finucane, K. C. Lee, M. P. Marcus, and H. Kress-Gazit, "Sorry dave, i'm afraid i can't do that: Explaining unachievable robot tasks using natural language." in *Robotics: Science and Systems*, 2013.

[12] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "Translating structured english to robot controllers," *Advanced Robotics*, vol. 22, no. 12, pp. 1343–1359, 2008.

[13] I. Chung, O. Propp, M. R. Walter, and T. M. Howard, "On the performance of hierarchical distributed correspondence graphs for efficient symbol grounding of robot instructions," in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*. IEEE, 2015, pp. 5247–5252.

[14] C. Matuszek, E. Herbst, L. Zettlemoyer, and D. Fox, "Learning to parse natural language commands to a robot control system," in *Experimental Robotics*. Springer, 2013, pp. 403–415.

[15] Y. Artzi and L. Zettlemoyer, "Weakly supervised learning of semantic parsers for mapping instructions to actions," *Transactions of the Association for Computational Linguistics*, vol. 1, pp. 49–62, 2013.

[16] J. Thomason, S. Zhang, R. Mooney, and P. Stone, "Learning to interpret natural language commands through human-robot dialog," 2015.

[17] L. P. Kaelbling and T. Lozano-Pérez, "Hierarchical task and motion planning in the now," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 1470–1477.

[18] S. Maniatopoulos, M. Blair, C. Finucane, and H. Kress-Gazit, "Open-world mission specification for reactive robots," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE, 2014, pp. 4328–4334.

[19] N. Piterman, A. Pnueli, and Y. Saar, "Synthesis of reactive (1) designs," in *Verification, Model Checking, and Abstract Interpretation*. Springer, 2006, pp. 364–380.

[20] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas, "Temporal-logic-based reactive mission and motion planning," *Robotics, IEEE Transactions on*, vol. 25, no. 6, pp. 1370–1381, 2009.

[21] U. Kuter, D. Nau, M. Pistore, and P. Traverso, "A hierarchical task-network planner based on symbolic model checking," in *ICAPS*, 2005, pp. 300–309.

[22] B. Akgun, M. Cakmak, K. Jiang, and A. L. Thomaz, "Keyframe-based learning from demonstration," *International Journal of Social Robotics*, vol. 4, no. 4, pp. 343–355, 2012.

[23] B. G. Weber, M. Mateas, and A. Jhala, "Learning from demonstration for goal-driven autonomy." in *AAAI*, 2012.