Robot-Initiated Specification Repair through Grounded Language Interaction

Adrian Boteanu[1], **Jacob Arkin**[2], **Siddharth Patki**[2], **Thomas Howard**[2] **and Hadas Kress-Gazit**[1] [*] [†] [‡]

Abstract

Robots are required to execute increasingly complex instructions in dynamic environments, which can lead to a disconnect between the user's intent and the robot's representation of the instructions. In this paper we present a natural language instruction grounding framework which uses formal synthesis to enable the robot to identify necessary environment assumptions for the task to be successful. These assumptions are then expressed via natural language questions referencing objects in the environment. The user is prompted to confirm or reject the assumption. We demonstrate our approach on two tabletop pick-and-place tasks.

INTRODUCTION

Establishing links between symbols such as language and their physical manifestation, a process known as symbol grounding (Harnad 1990), is a central problem in natural language understanding for human-robot interaction (HRI) (Stubbs, Hinds, and Wettergreen 2007; Lemaignan et al. 2012). Grounded language enables robots to interpret and respond to instructions, allowing information exchange about the physical world between humans and robots. The outcome of grounding affects the robot autonomy (Goodrich and Schultz 2007).

As robots become more autonomous and perform increasingly complex tasks, more descriptive groundings are needed in order to represent these tasks. This representation problem is further complicated by possibly incomplete or ambiguous natural language instructions (NLI), for example excluding elements that users consider self-evident. This creates the potential for the robot to incorrectly interpreting user intent. In particular, complex instructions consisting of multiple interdependent steps propagate these errors: the sorting instruction in Figure 1 implies performing a "pick up" action on "the blue cube," followed by a "place" action "in the right bin." The success of the drop action depends on first correctly executing the pick-up.

One existing method of grounding NLI employs probabilistic graphical models, which interpret single instructions (e.g. "pick up the crate") by mapping phrases to objects and actions (Tellex et al. 2011). An evolution of this approach, the Distributed Corespondence Graph (DCG), learns symbolic constraints to limit the search space of the grounding problem, allowing the use of large symbolic representations for grounding (Howard, Tellex, and Roy 2014). A limitation of these models is that the task plan produced through grounding is not assessed for correctness, potentially leading to inconsistencies and errors that propagate along multiple execution steps.

Physical groundings have been complemented with verifiable logical formluae in order to enable robots to execute complex instructions of conditionally dependent steps (Boteanu et al. 2016). The Verifiable-DCG model (V-DCG) extends the DCG symbolic representation to include both physical objects and Linear Temporal Logic (LTL) formulae. Grounding instructions using V-DCG generates formal LTL specifications which are then synthesized into verified controllers. Successful synthesis guarantees that the high-level symbolic task is achievable. Conversely, synthesis failures reveal logical conflicts in the specification that make it impossible to execute in all or some environments defined in the specification.

If synthesis is unsuccessful (i.e. the specification is *unsynthesizable*), the robot player cannot achieve its task. Unsynthesizable specifications are either *unsatisfiable* or *unrealizable*. The robot is unable to follow *unsatisfiable* specifications in any environment; for example in the sorting environment shown in Figure 4(a), an unsatisfiable specification would require the robot to both pick up a red block with its left gripper and simultaneously to never pick up with any of its grippers. Unrealizable specifications only allow the robot to respond to environment changes for a finite number of steps, after which the robot is unable to continue executing its task. Unrealizable specifications imply that there exists at least an *admissible* environment in which the robot is unable to achieve its task (Raman and Kress-Gazit 2013). An

[*] [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.

[†] [2]Jacob Arkin (`j.arkin@rochester.edu`), Siddharth Patki (`spatki@ur.rochester.edu`) and Thomas Howard (`thoward@cs.rochester.edu`) are with the Hajim School of Engineering and Applied Sciences, University of Rochester, Rochester, NY.

[‡]This work was supported by NSF NRI award 1427030.

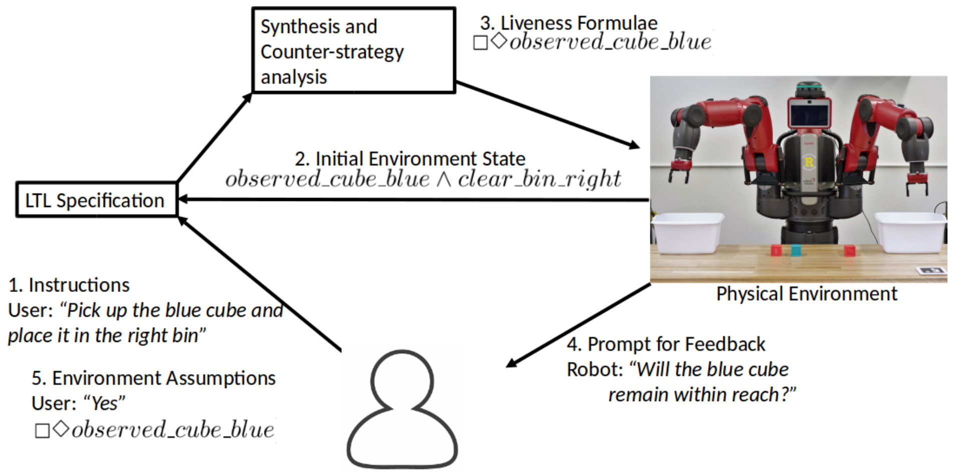
Synthesis and Counter-strategy analysis
3. Liveness Formulae
$\square\diamond observed\_cube\_blue$
2. Initial Environment State
$observed\_cube\_blue \wedge clear\_bin\_right$
LTL Specification
Physical Environment
1. Instructions
User: "Pick up the blue cube and place it in the right bin"
4. Prompt for Feedback
Robot: "Will the blue cube remain within reach?"
5. Environment Assumptions
User: "Yes"
$\square\diamond observed\_cube\_blue$

Figure 1: User interaction diagram showing main steps of our approach. Completing grounded specifications with environment assumptions is an iterative process which begins with a user command, which the system grounds to an LTL specification. The specification is then analyzed and assumptions are derived from it. The user is then prompted to confirm or reject these assumptions.

admissible environment is a state permitted by the specification which can be reached in a finite number of steps starting from one of the possible initial environment states.

The V-DCG model is able to identify unsatisfiable specifications obtained from grounding NLI, highlighting errors in the task description (e.g. requesting a drop action without a corresponding pickup). In this paper we contribute the following by building on the V-DCG model:

- We expand the V-DCG grounding symbol hierarchy to allow richer representations;
- We provide a method for situated natural language interaction between the robot and the user, in which the robot can respond to NLI from the user by requesting approval for environment assumptions. Our approach has the following benefits: (1) the user can provide assumptions without expert knowledge about the robot's capabilities; (2) the system is proactive and initiates user interaction as needed, guiding the user in providing necessary assumptions; (3) the final specification more accurately represents the execution environment, since its assumptions are established through situated interaction.
- We demonstrate our model on a significantly larger instruction corpus (two tasks and 60 instructions, from one task and six instructions (Boteanu et al. 2016)).

The core contribution of our paper, obtaining environment assumptions through user interaction, is an iterative process (Figure 1):

1. The user states an instruction which is grounded to a LTL specification using a pre-trained model. The system then attempts to synthesize the LTL specification into a controller. If the specification is synthesizable, execution begins. If it is unsatisfiable, existing work can inform the user about possible causes (Raman et al. 2013; Boteanu et al. 2016). Our contribution targets unrealizable outcomes, in which some admissible environments can impede the robot, thus the following assumes the initial specification is unrealizable;
2. We supplement the specification with the environment's initial state as perceived by the robot, and reattempt synthesis;
3. If the specification is again unrealizable, we extract the collection of environment moves that will make the robot fail, which are known as *counter-strategy*. An unrealizable specification can be executed for a finite number of steps until the robot is unable to respond to environment changes. These are worst-case scenarios which are possible given the current specification. While for unrealizable specifications there may also be environments in which the robot can successfully execute its goals indefinitely, one advantage of our approach to use formal verification in generating controllers is that we obtain a worst-case analysis of all possible execution sequences. We use existing work in counter-strategy analysis to generate environment assumption formulae that would, if added to the specification, produce a synthesizable specification (Alur, Moarref, and Topcu 2013);
4. We use the proposition evaluation results and the assumption formulae derived from the counter-strategy to generate prompts, which the user can either accept or reject;
5. If the user answers a prompt affirmatively, we add the corresponding formulae to the specification. We then attempt to re-synthesize the specification, continuing steps 3, 4 and 5 until synthesis succeeds or the user denies all prompts generated by the system. Once synthesis has succeeded, the robot executes the task using the resulting reactive controller.

The rest of the paper is structured as follows: we first review relevant literature, and summarize the LTL formalism and the V-DCG model. We then describe our contribution in generating environment assumptions and feedback prompts. Finally, we evaluate our grounding model on two tasks: a binary cube sorting task and a cube stacking task, for which we crowdsourced and annotated an instruction corpus.

RELATED WORK

Natural language grounding and natural language interaction have been studied in HRI from a broad range of perspectives ranging from psychological motivations to enabling robot autonomy (Roy 2005). To the best of our knowledge, our work is the first to incorporate stochastic language grounding, robot-driven language interaction for task repair, and generating language from formal synthesis results. We will now summarize existing work in these areas.

One approach to grounding NLI uses probabilistic models which map the meaning of parsed language to motion-planned trajectories, regions in physical space and objects (Generalized Grounding Graph, G^3) (Tellex et al. 2011). The DCG model extends the G^3 model by defining symbolic constraints which partition physical space prior to planning (Howard, Tellex, and Roy 2014). A central feature of the DCG model is that, in addition to physical groundings such as objects and trajectories, the grounding can use arbitrary symbolic representations. For example, the original model used spatial symbols to describe concepts such as *near*, *right* and *left*, while recent contributions have introduced abstract groupings such as rows of objects (Paul et al. 2016). In these approaches, individual grounding results are evaluated together over instruction sequences, which can lead to inconsistencies between different instructions as a result of the generative nature of these models. The V-DCG model introduces logical groundings which allow instruction sequences to be holistically verified (Boteanu et al. 2016).

Human-robot interaction for grounding has been shown in robot learning from demonstration (Argall et al. 2009). When interpreting instructions, the robot would recursively prompt the user to clarify unknown groundings (Lauriar et al. 2001; Chao, Cakmak, and Thomaz 2011). Stochastic grounding models have been used to generate salient object references to allow the robot to request help from users, enabling recovery from execution exceptions that the robot cannot address by itself (Tellex et al. 2014). Voice prompts have also been used to enable users to help robots adapt known tasks to new environments by requesting the user to verify substitutes proposed by the robot (Boteanu et al. 2015).

Existing work that leveraged formal synthesis to generate verbal feedback (Raman et al. 2013) relied on manually-defined groundings for actions, whereas our model learns these symbols. Other work in formally representing robot instructions uses Combinatorial Categorical Grammars to infer logical representations of navigation instructions (Matuszek et al. 2013).

PRELIMINARIES

Linear Temporal Logic Overview

Linear Temporal Logic (LTL) is defined over propositions and formulae taking Boolean values over an infinite discrete time series. Propositions are atomic variables which hold a binary truth value at each time step and offer discrete abstractions: *Sensor* propositions, abstracting the *environment*, $\{x_i | i = 1..n\}$, become active when a perception condition is met, for example an object is recognized; *Action* propositions, abstracting the *robot*, $\{\alpha_i | i = 1..m\}$, command actuation primitives such as navigation and grasping. In addition, the system can represent *memory* propositions, which correspond to internal states. Formulae, ϕ, are formed by applying Boolean ($\neg, \wedge, \vee$) and temporal operators (next, $\bigcirc$, until, U, together with the derived symbols eventually, $\diamond$ and always, $\square$) to proposition and other formulae. The temporal operators have the following semantics: $\bigcirc\phi$ – at the next time step the ϕ will be True; $\psi U \phi$ – at some future time step ϕ will be True, and ψ must be true until then; $\diamond\phi$ – at some future time step, ϕ will be true; $\square\phi$ – ϕ is True for every time step. For example, $\square\diamond\phi$ means that the formula will repeatedly become *True*; $\square(x_i \rightarrow \alpha_j)$ expresses that at all time steps the logical implication holds.

We use the GR(1) fragment of LTL since it offers tractable synthesis (Piterman, Pnueli, and Saar 2006). GR(1) formulae follow an assume-guarantee structure describing a two-player game in which the robot player responds through actions to changes in sensors controlled by an adversarial environment player; the robot's behavior is thus reactive to the environment. GR(1) specifications abstract tasks as a two-player game: the *robot* and the *environment* in which the robot operates, both operating under formal assumptions described through LTL formulae (Kress-Gazit, Fainekos, and Pappas 2008). The robot player has control over *action propositions*, which interface with actuation, while the environment player controls *sensor* propositions, which correspond to the robot's perception. Both players' behaviors are restricted through *assumptions*. Assumptions restricting the behavior of either player are considered *safeties*, while *liveness* are assumptions that express goals. For the example in Figure 1, an environment assumption states that a block will eventually appear in the environment, and the robot can only perform a pick up action if it senses a block. The specification defines an initial state for the game, ϕ_i. If successful, synthesis produces a controller that guarantees the robot's behavior under environment assumptions (Kress-Gazit, Fainekos, and Pappas 2009).

V-DCG Overview

The V-DCG grounding model generates LTL specifications by grounding natural language using the following symbol hierarchy:

- *Object*(*id*, T_o, C_o), where *id* is a unique object identifier produced by the perception system; T_o is a type from a known set of object types, for example {*cube, bin, robot gripper*}; C_o is a color from a given set, for example {*blue, red, green*}. These properties can be easily extended by adding different attributes and possible values for each attribute;
- *Sensor*(T_S, x_i, g_o), where T_S denotes a sensor type which indicates which robot perception primitive should be invoked when evaluating this proposition; x_i is the proposition as it is used in the LTL specification; g_o are

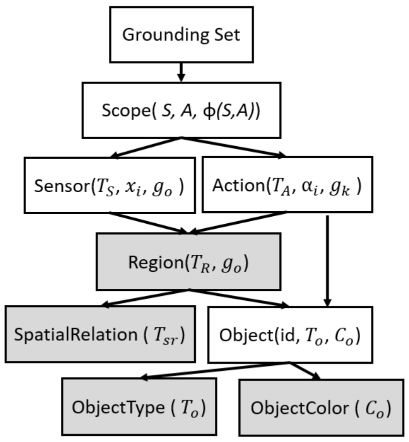

Figure 2: We expand the symbolic hierarchy of the V-DCG model by adding *ObjectType*, *ObjectColor*, *SpatialRelation* and *Region* symbols (in grey).

grounded *Object* types associated with this sensor;

- *Action*(T_A, α_i, g_k), where T_A indicates the robot's actuation primitive to be invoked when activating the proposition corresponding to this action; α_i is the proposition used in the LTL specification; g_k are grounded objects which are used by the actuation function T_A;
- *Scope*($S, A, \phi(S, A)$) combines sensors, S, and action A. Depending on the types of sensors and actions, different formulae $\phi(S, A)$ are produced. This mapping is defined as part of the grounding model;
- *Grounding Set* is the union of all other grounding symbols inferred from a parsed instruction; instructions are parsed using a CYK parser (Gonzalez and Thomason 1978) and a production grammar to annotate the sentence with Penn TreeBank tags (Marcus et al. 1994). The specification conjuncts all LTL formulae $\phi(S, A)$ present in *Scopes* of the *Grounding Set*.

The V-DCG model assumes that the environment *safety* formulae defined during model training hold for new grounding environments. If the environment exhibits behaviors not captured by these formulae and violates the specification, execution is halted. In this paper we enable specifications generated from grounding NLI in new environments to be interactively supplemented by the user, such that any necessary assumptions that were not captured during model training are added before execution.

SYMBOLIC REPRESENTATION

We extend the V-DCG symbol hierarchy with new symbols (Figure 2). Firstly, we enable the model to better learn from ambiguous object references by leveraging spatial information relative to the robot. For example, an ambiguous phrase such as "the box" in an environment with multiple cubes (Figure 3) can be clarified by the spatial reference "on the right". To do so, we added the symbols:

- *ObjectType*, which have the same types as *Objects*, T_o;
- *ObjectColor*, with C_o the same as for *Objects*;

```
<PP><Region type="inside">
    <Object id="bluecube2"
            type="cube" color="blue"
            position="0.6,-0.4,0.8"/>
    </Region>
    <IN text="into"/>
    <NP><Object id="bluecube2"
            type="cube" color="blue"
            position="0.6,-0.4,0.8"/>
      <NP>  <ObjectType type="cube"/>
            <DT text="the"/>
            <NN text="box"/></NP>
      <PP>  <SpatialRelation type="right"/>
            <IN text="on"/>
            <NP> <SpatialRelation type="right"/>
              <DT text="the"/>
              <NN text="right"/> </NP>
</PP></NP></NP></PP>
```

Figure 3: Annotation examples showing symbols introduced in this work which expand the V-DCG representation: *Object Type*, *Spatial Relation* and *Region*.

- *SpatialRelation*$\in${*center, left, right, above*}.

Secondly, in order to correctly associate actions and objects, we introduce *Region* symbols. *Regions* associate *Objects* with prepositions (e.g. "*on* the red block"), thus restricting the space of possible actions that can be performed on that object.

Figure 3 shows an example of how these symbols are used to ground the phrase *"into the box on the right"*. First, an *ObjectType* symbol of type *cube* is associated with the Noun Phrase (NP) *the box*, then *SpatialRelation* symbols are associated with the prepositional phrase *on the right* and the noun phrase *the right*. These symbols are combined in a NP to identify an *Object* of type *cube*, with known color and position. A *Region* is associated with the preposition "into" and is used to describe the Prepositional Phrase (PP) representing the phrase.

The final LTL formula associated with a scope is constructed by mapping proposition types and their objects to formula templates which establish conditional dependence between the propositions and are determined by the task type. We will give two examples, in which *observed_cube_blue* and *understack_cube_red* are sensor propositions, *pickup_right* is an action proposition, and *right_gripper* is a memory proposition:

The phrase *"Pick up the blue cube with your right hand"* for the sorting task is grounded to the formulae:

- $\square((\neg \bigcirc observed\_cube\_blue) \vee right\_gripper) \rightarrow \neg \bigcirc pickup\_right)$ (1)
- $\square\Diamond(pickup\_right)$ (2)

Formula 1 states that the action proposition *pickup_right* can not be activated if *observed_cube_blue* is *False* and *right_gripper* is *True*, i.e. the robot cannot pick up if it doesn't see a blue cube or its gripper is full. Formula 2 states the goal of picking up. The phrase *"Take the red cube"* for the stacking task is grounded to the formulae, which state that the robot cannot pick up with its right gripper if there is no red cube observed or the cube is under a stack, or the gripper is full, together with the goal of picking up with the right gripper:

- $\square((\neg \bigcirc observed\_cube\_red) \vee understack\_cube\_red \vee$

$right\_gripper) \rightarrow \neg \bigcirc pickup\_right)$ (3)
- $\square\diamond(pickup\_right)$ (4)

We defined additional features for training the DCG log-linear model which make use of these additional symbols. As in the DCG model, the likelihood of a candidate symbol is computed using *features* that implement specific heuristics depending on the symbols, language and physical environment. The model learns bottom-up symbol structures over a tagged parse tree, such as the one shown in Figure 3. The model defines a *symbol space* containing all symbol types that it can express, such as objects of all possible types and colors and scopes of all possible sensors and actions and objects. When training, features are evaluated bottom-up starting from leaf-nodes in the parse tree (i.e. surface words). Features match part-of-speech and symbols at the current level with evidence from lower levels of the tree (both parsed language and symbols), and count as positive or negative evidence in the model, changing the likelihood of symbols corresponding to a sub-tree. When predicting groundings for a new parse tree, candidate symbols that ground to the parse tree up to that point are evaluated against the language and the underlying symbols, and then the most likely *N* are propagated up the tree. (for all experiments shown in this paper *N=4*).

Thus, first the word terminal contained in the leaf nodes will be associated to a grounding. In the above example, the words "the box" are grounded to an *ObjectType* symbol. As the learner moves up in the hierarchy of the tree, each level needs to match or compose symbols from its children. For example, to infer the *Object* grounding of the phrase, the *SpatialRelation* and *Object* symbols must pair accordingly. For the *SpatialRelation* the learner checks if the *Object* coordinates correspond to its type, while for *ObjectType* and *ObjectColor* it matches the object's type and color, respectively. The learning process continues similarly for higher-level symbols; the meaning of the entire instruction is a union of all *Scope* symbols.

We note that the model can be easily extended to ground more complex object references (e.g. nested references) by adding features that interpret such langauge and associate it with symbols (Chung et al. 2015).

ENVIRONMENT ASSUMPTION FEEDBACK

Using the previously described symbol hierarchy, our model generates LTL specifications by grounding instructions from the user. We present two complementary contributions for repairing this specification if it is unrealizable: (1) by including formulae describing the initial environment as perceived by the robot; (2) by including liveness formulae describing environment assumptions approved by the user.

To generate a formula describing the environment's initial state, we evaluate all sensor functions on the current environment and add to the specification a formula expressing the environment's initial state. For example, the environment shown in Figure 4(a) would have the initial assumptions $(observed\_cube\_blue \wedge right\_bin\_clear \wedge observed\_cube\_red \wedge left\_bin\_clear)$.

If the specification is realizable after adding the initial environment formulae, the robot proceeds with executing the synthesized controller and does not prompt the user.

For an unrealizable specification, transitions in the counter-strategy can be used to identify formulae which would limit the environment and allow synthesis (Alur, Moarref, and Topcu 2013). To obtain an environment safety formula, $\phi_e(X_i)$, we use a simplified implementation of this approach, tailored to generating assumptions of the form $\square\diamond(\wedge[\neg](X_i)), i = 1..n$, i.e. a conjunction where $X_i, i = 1..n$ are sensor propositions that can be negated. This allows us to translate these formulae to binary question prompts for the user.

Formulae (1) and (2) require a single-sensor assumption, $\square\diamond observed\_cube\_blue$, while formulae (3) and (4) require a two-sensor assumptions, $\square\diamond observed\_cube\_red \wedge \neg understack\_cube\_red$. Our interaction method is not limited to these templates and can be expanded to more complex formulae provided an appropriate prompting method is chosen.

CONVEYING ASSUMPTIONS THROUGH NATURAL LANGUAGE

Our design goal for generating prompts was to convey assumptions succinctly in order to minimize the burden on the user. We generate prompts that reference objects in the physical environment, which the user can accept or reject. We express an environment liveness formula, $\phi_e(X_i)$, where $X_i, i = 1..n$ are sensor propositions of type $T_S(i)$ grounded to object $g_o(i)$, through natural language by using a template specific to the sensor type. This template is filled in using the language uttered by the user to reference the object targetted by the sensor proposition. These templates are filled in with an object reference, shown in italics in the following examples, which correspond to environments shown in Figure 4:

- Figure 4(a), one sensor: "Will *the blue block* remain within reach?"
- Figure 4(b), two sensors: "Will *red cube* remain within reach and will you remove *the block on top of red cube*?"

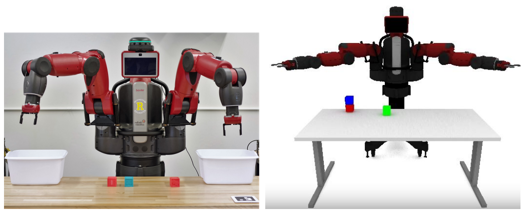

Figure 4: Physical and simulated grounding environment examples: (a) sorting red and blue cubes, both bins clear; (b) stacking the red cube on the green cube is blocked by the blue cube.

EXPERIMENTAL RESULTS

We evaluated the learning performance our model on two tabletop object manipulation tasks using a Rethink Robotics Baxter Research Robot, for which we designed different specifications in LTL:

- **Cube sorting:** Blue and red were to be sorted into bins placed on either side of the robot; the robot could only pick up a cube if its gripper was empty; the bins could be covered by a lid at any time, preventing cubes from being placed into them (Figures 4(a)).
- **Cube stacking:** The robot had to pick up a cube of a given color (red, green or blue) and stack it on top of another cube; the robot could only pick up a block if it was not under another cube stacked on top of it (Figure 4(b)); similarly, stacking was allowed only if the destination block was not obstructed (Figure 4(b)).

We trained a V-DCG grounding model following a procedure similar to (Boteanu et al. 2016): collect NLI using crowdsourcing, annotate a training corpus, and design features for training a model using the corpus. To collect the training corpus, we executed each specification in eight different simulated environments with varying numbers of blocks and bin behaviors (16 environments in total for the two tasks). For each environment we used manually defined groundings for the propositions, synthesized a controller, and rendered videos of a simulated Baxter executing the task.

We then crowdsourced NLI through surveys[1] that showed videos of the robot's execution and required users to freely type instructions that would produce that behavior. The responses consisted of multi-sentence instructions e.g. "Pick up the innermost blue cube and drop into the right box. Then pick up the remaining blue cube and the innermost red cube. Drop the red cube into the box on the left and then pick up the other red cube. Drop it into the left box and drop the blue one into the right."

From the collected instructions we selected two corpora of multi-sentence instructions: 39 instructions for the sorting task and 21 instructions for the stacking task. We separated answers into sentences, parsed them using a production grammar and annotated the parsed instructions using the symbol hierarchy shown in Figure 2. The following are individual training instructions from the annotated training corpus:

Sorting:
"Pick up the remaining red cube and drop it in the left bin;"
"Pick up the blue block and put it in the box on your right;"
"Put the blue cube in the right bin and put the red cube in the left bin."

Stacking:
"Take the red cube. Put the red cube on the green cube;"
"Take a green cube. Wait to remove blue cube. Put green cube on red cube."
"Take the blue cube, lift the cube, then place it above the green cube."

[1]We deployed identical surveys on http://www.mturk.com for the sorting task, and on http://www.crowdflower.com for the stacking task.

The two-task corpus of 60 instructions (21 stacking, 39 for sorting) contained 828 words (158 unique words) and expressed the following numbers of individual symbols: 153 ObjectType, 109 ObjectColor, 588 Object, 51 SpatialRelation, 60 Region, 275 Sensor, 223 Action, and 222 Scope. Our model covered (i.e. features could discriminate and represent annotations) 99.9872% of all the 1,132,710 factors in the joint corpus.

We evaluated V-DCG on the two-task corpus. Training and testing on all 60 instructions, the model could recover 42 instructions (17 for the stacking task and 25 for the sorting task). Training and testing separately, the model fully inferred 19/21 stacking instructions and 28/39 sorting instructions. Given that the current features offer good coverage of the factor space, as we expand these corpora additional data will improve performance on inferring full instructions. The annotations between the two tasks have significant lexical, grammatical, and symbolic grounding differences; for example, "green" is only used in the stacking task, and "bin" is only used in the sorting task. However, the two tasks contain similar phrases referring to red or blue blocks. The current performance on the joint corpus is slightly lower than individually training models for each task due to an increase in size of the symbol space.

We deployed the trained model on a Rethink Robotics Research Baxter robot and implemented the interaction behavior shown in Figure 1 using open-source speech recognition[2] and text-to-speech[3] software. The robot uses tabletop segmentation of RGBD point-clouds to determine properties of the cubes on the table. The location of bins are pre-determined. A sampling-based motion planner was used to determine arm trajectories. Grasping of cubes on the final approach is performed by a controller using visual-servoing. Sample instructions are available in the supplemental video[4].

CONCLUSION

In this paper we introduce a natural language interaction method which enables the robot to request the user to approve or reject automatically-generated environment assumptions. The interaction is triggered by synthesis outcomes which identify environment behaviors that would prevent the robot from accomplishing the task. LTL assumption formulae that restrict the environment from exhibiting these behaviors are generated automatically and then expressed through natural language in prompts which the user can accept or reject.

Grounding complex NLI to the physical environment and LTL formulae enables the synthesis of guaranteed controllers that can accomplish the task described in the instruction. Synthesis guarantees robot behavior under assumptions describing constraints of the robot and the environment. If a grounding model is used in new environments, our contribution can identify new assumptions that will be needed.

[2]http://cmusphinx.sourceforge.net
[3]http://espeak.sourceforge.net/
[4]https://youtu.be/sG8dNGiyNnM

References

Alur, R.; Moarref, S.; and Topcu, U. 2013. Counter-strategy guided refinement of gr (1) temporal logic specifications. In *Formal Methods in Computer-Aided Design (FMCAD), 2013*, 26–33. IEEE.

Argall, B. D.; Chernova, S.; Veloso, M.; and Browning, B. 2009. A survey of robot learning from demonstration. *Robotics and autonomous systems* 57(5):469–483.

Boteanu, A.; Kent, D.; Mohseni-Kabir, A.; Rich, C.; and Chernova, S. 2015. Towards robot adaptability in new situations. In *2015 AAAI Fall Symposium Series*.

Boteanu, A.; Howard, T.; Arkin, J.; and Kress-Gazit, H. 2016. A model for verifiable grounding and execution of complex natural language instructions. In *IEEE IROS 2016, Proceedings of*.

Chao, C.; Cakmak, M.; and Thomaz, A. L. 2011. Towards grounding concepts for transfer in goal learning from demonstration. In *2011 IEEE International Conference on Development and Learning (ICDL)*, volume 2, 1–6. IEEE.

Chung, I.; Propp, O.; Walter, M. R.; and Howard, T. M. 2015. 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*, 5247–5252. IEEE.

Gonzalez, R. C., and Thomason, M. G. 1978. Syntactic pattern recognition: An introduction.

Goodrich, M. A., and Schultz, A. C. 2007. Human-robot interaction: a survey. *Foundations and trends in human-computer interaction* 1(3):203–275.

Harnad, S. 1990. The symbol grounding problem. *Physica D: Nonlinear Phenomena* 42(1):335–346.

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

Kress-Gazit, H.; Fainekos, G. E.; and Pappas, G. J. 2008. Translating structured english to robot controllers. *Advanced Robotics* 22(12):1343–1359.

Kress-Gazit, H.; Fainekos, G. E.; and Pappas, G. J. 2009. Temporal-logic-based reactive mission and motion planning. *Robotics, IEEE Transactions on* 25(6):1370–1381.

Lauriar, S.; Bugmann, G.; Kyriacou, T.; Bos, J.; and Klein, E. 2001. Training personal robots using natural language instruction. *IEEE Intelligent systems* (5):38–45.

Lemaignan, S.; Ros, R.; Sisbot, E. A.; Alami, R.; and Beetz, M. 2012. Grounding the interaction: Anchoring situated discourse in everyday human-robot interaction. *International Journal of Social Robotics* 4(2):181–199.

Marcus, M.; Kim, G.; Marcinkiewicz, M. A.; MacIntyre, R.; Bies, A.; Ferguson, M.; Katz, K.; and Schasberger, B. 1994. The penn treebank: annotating predicate argument structure. In *Proceedings of the workshop on Human Language Technology*, 114–119. Association for Computational Linguistics.

Matuszek, C.; Herbst, E.; Zettlemoyer, L.; and Fox, D. 2013. Learning to parse natural language commands to a robot control system. In *Experimental Robotics*, 403–415. Springer.

Paul, R.; Arkin, J.; Roy, N.; and Howard, T. M. 2016. Efficient grounding of abstract spatial concepts for natural language interaction with robot manipulators. In *Proceedings of Robotics: Science and Systems*.

Piterman, N.; Pnueli, A.; and Saar, Y. 2006. Synthesis of reactive (1) designs. In *Verification, Model Checking, and Abstract Interpretation*, 364–380. Springer.

Raman, V., and Kress-Gazit, H. 2013. Explaining impossible high-level robot behaviors. *IEEE Transactions on Robotics* 29(1):94–104.

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

Roy, D. 2005. Grounding words in perception and action: computational insights. *Trends in cognitive sciences* 9(8):389–396.

Stubbs, K.; Hinds, P. J.; and Wettergreen, D. 2007. Autonomy and common ground in human-robot interaction: A field study. *IEEE Intelligent Systems* 22(2):42–50.

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

Tellex, S.; Knepper, R.; Li, A.; Rus, D.; and Roy, N. 2014. Asking for help using inverse semantics. *Proceedings of Robotics: Science and Systems, Berkeley, USA*.