Towards Problem Solving Agents that Communicate and Learn

Anjali Narayan-Chen[1], **Colin Graber**[1], **Mayukh Das**[2], **Md Rakibul Islam**[3],
Soham Dan[1], **Sriraam Natarajan**[2], **Janardhan Rao Doppa**[3],
Julia Hockenmaier[1,5], **Martha Palmer**[4] and **Dan Roth**[1]
[1]University of Illinois at Urbana-Champaign, [2]Indiana University, [3]Washington State University,
[4]University of Colorado at Boulder, [5]Allen Institute for Artificial Intelligence

Abstract

Agents that communicate back and forth with humans to help them execute non-linguistic tasks are a long sought goal of AI. These agents need to translate between utterances and actionable meaning representations that can be interpreted by task-specific problem solvers in a context-dependent manner. They should also be able to learn such actionable interpretations for new predicates on the fly. We define an agent architecture for this scenario and present a series of experiments in the Blocks World domain that illustrate how our architecture supports language learning and problem solving in this domain.

1 Introduction

An agent that can engage in natural, back-and-forth communication with humans to help them complete a real world task requires the ability to understand and produce language in the context of that task (i.e. to map between utterances and meaning representations the problem solving components of the agent can act on in a particular situation). The agent may also need to initiate clarification requests when communication fails, and to learn new domain (or conversation) specific vocabulary and its meaning. This kind of symmetric, grounded communication with a problem-solving agent goes significantly beyond the one-step, single direction understanding tasks considered in standard semantic parsing (e.g. Zelle and Mooney, 1996; Zettlemoyer and Collins, 2005; Clarke et al., 2010) or even short, simple instructions to robots (e.g. Tellex et al., 2011).

In order to focus on these concept learning and communication issues, we deliberately limit ourselves here to a simple, simulated environment. We operate in a two-dimensional Blocks World domain where a human wants one or more shapes to be constructed on a grid. The human needs to communicate the goal of this planning task to the agent. Once the agent has understood the instructions and its planning is done, it communicates its plan to (possibly) another human who will then execute this plan. Depending on the complexity of the task and the linguistic capabilities of the agent, this scenario may require a lot of back-and-forth communication. If the human omits details from their description and prevents the agent from accomplishing the task, we expect the agent to initiate communication and ask clarification questions. If the human uses vocabulary that is new to the agent, we expect the agent to ask for a definition and the human to teach the agent its meaning in the domain.

We define an agent architecture named COG that allows us to investigate the challenges arising in this symmetric communication scenario. COG combines a problem solving (planning) component with a basic language understanding and generation system that is initially only equipped with a limited vocabulary. We perform a sequence of experiments of increasing complexity that illustrate how our architecture supports the problem solving scenarios described above, and how language learning is accomplished within this architecture. We argue that, within this architecture, all the agent's capabilities – comprehension, production, problem solving and learning – can be improved with additional communication.

Section 2 defines the domain and problem setup. Section 3 provides an overview of the COG architecture. Section 4 describes COG's current components (language comprehension, production, memory, problem solving, and dialogue mediation). Section 5 describes our experiments and results.

Proceedings of the First Workshop on Language Grounding for Robotics, pages 95–103,
Vancouver, Canada, July 30 - August 4, 2017. ©2017 Association for Computational Linguistics

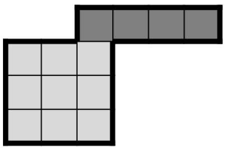

Figure 1: A complex shape, which can be viewed as conjunction of simpler known shapes: a row (dark, `width` = 4) and a square (light, `size` = 3).

2 Domain and Problem Setup

We consider a two-dimensional (2D) Blocksworld domain. There is a 2D grid and the goal is to build different configurations with square blocks based on their natural language descriptions. We assume agents come equipped with the vocabulary and definitions of primitive concepts, e.g. for simple shapes (*row*, *square*, etc.) and spatial relations (*adjacent*, *on top of*, etc.), but may have to learn the definitions of new terms and concepts that arise in communication with the human user.

We define three different types of goal descriptions and design corresponding tasks for evaluation. During evaluation, the agent must automatically identify the task and respond appropriately.

Task 1: Complete descriptions. For the first task, the agent is provided with a complete description, in natural language, of a target configuration consisting of one or two shapes. The definition is complete and does not require further clarification. This tests the ability of the agent to understand and ground specific descriptions.

Task 2: Descriptions with missing information. For the second task, the agent is again provided with a description of a target configuration consisting of one or two shapes; however, the description will not be specific enough for the agent to be able to ground it to a unique configuration. This tests the ability of the agent to recognize when some information is missing from a description and to initiate a dialogue which will clarify these details.

Task 3: Descriptions of new shapes. For the third task, the agent is asked to construct a complex shape which is not contained within its initial vocabulary of primitive concepts (e.g., the letter "L"). This tests the ability of the agent to extend its vocabulary through interaction with a human.

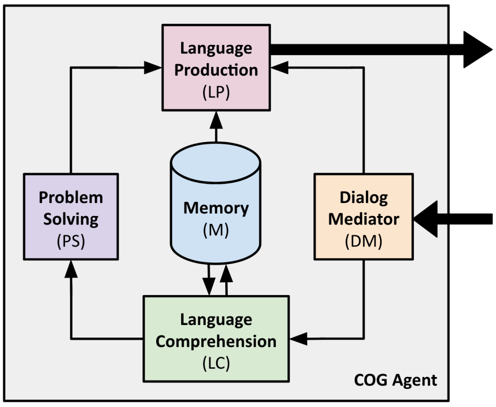

Figure 2: The agent architecture. Interaction (thick arrows) is done via the Dialogue Mediator (input) and the Language Production component (output). Thin arrows indicate the flow of information between the internal components.

3 The COG Agent Architecture

Agents that solve a task require explicit knowledge of the corresponding real world concepts. Agents that also communicate about solving tasks additionally need to be able to map linguistic expressions such as "square" or "on top of" to these concepts. They also need to know to which situations these concepts apply (e.g. whether a given configuration can be referred to as a square, or whether a row can be placed on top of that square). The representations used by the different components of our agent therefore vary in their specificity. This requires additional work to bridge the resulting representational gaps. To represent the definition of concepts, we store "lifted" representations, i.e. rules whose predicates only contain uninstantiated variables (e.g. rectangles have height and width). By contrast, problem solving requires fully grounded representations in which shapes are placed at specific locations on the grid (e.g., a 3 by 4 rectangle located at $(0, 0)$). Language comprehension and production operate on a middle ground where some parameters may be instantiated, even if shapes are not necessarily placed at specific locations (e.g., a 3 by 4 rectangle).

Figure 2 describes the architecture of our agent. The **Language Comprehension (LC)** module converts natural language into an executable, grounded, declarative representation in a STRIPS-like language (Fikes and Nilsson, 1971) that the **Problem Solving (PS)** module can act on. PS returns partial or complete problem solving plans, also in this language. If grounding fails, LC produces queries that are sent to the **Language Production (LP)** module.

The LP takes these queries or the plans produced by PS and returns natural language output. The **Memory (M)** module stores lifted representations of built-in and acquired predicates that are used by LC and LP. The **Dialogue Mediator (DM)** is aware of the current dialogue state. It can augment the natural language input before transferring it to LC (e.g. if the human utters a number, DM might provide the context, letting LC know that the number refers to the size of a specific shape). DM is also used by LP to generate utterances that are appropriate for the current dialogue state. Instead of narrating the fully grounded, block-wise sequential plans produced by PS, LP may identify subsequences of steps that correspond to instantiations of more abstract concepts (e.g., a sequence of blocks placed horizontally is a "row"), and generates language at this more natural level.

4 Current Components of COG

In this section, we describe the current implementations of the different modules (language comprehension, memory, problem solving, language production, and dialogue mediation) in COG, noting that the architecture is flexible and allows for us to plug-in other implementations as needed.

4.1 Language Comprehension

The LC module consists of semantic parsing and language grounding components.

4.1.1 Semantic Parsing

Our semantic parser is implemented as a neural sequence-to-sequence model with attention (Bahdanau et al., 2014; Dong and Lapata, 2016; Jia and Liang, 2016). The model consists of two LSTMs. The first LSTM (the encoder) processes the input sentence $\mathbf{x} = (x_1, \ldots, x_m)$ token-by-token, producing a sequence of hidden states $\mathbf{h}^s = (h^s_1, \ldots, h^s_m)$ as output. The second LSTM (the decoder) models a distribution $P(y_i|y_1, \ldots, y_{i-1}; h^s)$ at each time step over output tokens as a function of the encoder hidden states and the previous outputs. The final parse $\mathbf{y} = (y_1, \ldots, y_n)$ is obtained by selecting the token at each time step that maximizes this probability and feeding a learned embedding for it into the LSTM as part of the next input. Multiple sentence inputs are processed sentence-by-sentence, where the initial encoder hidden state for a given sentence is set to the final encoder hidden state for the previous sentence, and the initial encoder hidden state for the first sentence is the zero vector. The final logical form is the conjunction of the logical forms for each individual sentence.

The parser is trained with a small fixed-size vocabulary; however, to represent new shapes it needs to be able to output new predicates for shapes that it has not encountered during training. We accomplish this by using an attention-based copying mechanism (Jia and Liang, 2016; Gulcehre et al., 2016; Gu et al., 2016). At every time step, the decoder may either output a token from the training vocabulary or copy a word from the input sentence. Hence, when new shapes are encountered in the input, the parser is able to copy the shape name from the input sentence to define a new predicate.

4.1.2 Language Grounding

Grounding is beyond the capabilities of a semantic parser, which may interpret a sentence such as "Place a square of size 4 on top of the rectangle" without any *understanding* of the key predicates (what is a "square"?) or relations (what is "on-top-of"?). LC therefore includes a grounding component which converts the output of the semantic parser into executable grounded representations that can be directly used by PS. This component obtains definitions of predicates from M and uses a geometric reasoner to identify feasible shape placements. The reasoner assigns location coordinates to each shape relative to the best bounding box it can find for the entire configuration, given the grid boundaries. A full description and its grounding process are given in Figure 3.

Grounding succeeds immediately if the agent is given a complete goal description, as in Task 1. Grounding fails if the goal description is incomplete or unknown, as in Tasks 2 and 3. In these cases, a clarification query Q is issued and passed to LP.

Formally, a **complete goal description** of a target configuration is defined as a tuple $\mathcal{G} = \langle \{\langle s_i, id_i, \wedge_k d_i^{(k)} \rangle\}_{i \in S}, \wedge_{j \in [S \times S]} f_j \rangle$, where S is the list of shapes and s_i, id_i and $\wedge_k d_i^{(k)}$ are the shape name, identifier, and dimension attributes of shape i respectively. f encodes pair-wise

Human:	"Build a square of size 3. Then, construct a row of 4 blocks above the square. The left end of the row should be above the upper right corner of the square."
Parse:	$\texttt{square(a)} \wedge \texttt{size(a,3)} \wedge \texttt{row(b)}$ $\wedge\ \texttt{width(b,4)} \wedge \ldots$
Ground:	$\texttt{square(0,2,3)},\ \texttt{row(2,5,4)}$
Plan:	(putdown b1 0.0 0.0), (putdown b2 1.0 0.0), (putdown b3 2.0 0.0), ...

Figure 3: A fully-specified goal description for the configuration in Figure 1 and its path through the architecture. The geometric reasoner inferred that the bounding box's lower left corner is at $(0, 2)$, which is also the lower left corner of the square.

spatial relations between shapes. A complete goal description for the configuration in Figure 1 is $\langle\{\langle\texttt{row}, \texttt{a}, \texttt{width} = 4\rangle, \langle\texttt{square}, \texttt{b}, \texttt{size} = 3\rangle\}, \texttt{spatialRelation}(\texttt{a}, \texttt{b}, \texttt{upperRightCorner})\rangle$.

An **incomplete goal description** $\mathcal{G}^I$ is a goal description where the values of one or more dimensional or spatial attributes are missing: $\mathcal{G}^I = \mathcal{G} - x$, where $x = x_d \cup x_f$ (with $x_d \subseteq \{d_i\}_{i \in S}$ and $x_f \subseteq \{f\}$) is the missing information. In this case, a query Q asks for values of the missing dimensions or spatial relations (i.e., $Q = x$). An incomplete goal description for Figure 1 is $\langle\{\langle\texttt{row}, \texttt{a}, \texttt{width} = \textbf{null}\rangle, \langle\texttt{square}, \texttt{b}, \texttt{size} = 3\rangle\}, \textbf{null}\rangle$. Here, the width of the row and the spatial relation between the shapes are unknown.

An **unknown goal description** $\mathcal{G}^U$ occurs when one or more of the shapes in a goal description are not known to the agent (i.e., the memory module M does not contain their definitions): $\mathcal{G}^U = \mathcal{G}$ if $\exists s_i$: $s_i \in S, s_i \notin$ M. In this case, a query Q asks for additional clarification about the unknown concepts (i.e., $Q = \mathit{Define}(s_i)$). Figure 1 could be described with a new concept $\texttt{Foo} \notin$ M: $\mathcal{G} = \texttt{Foo(p)} \wedge \texttt{dim}_1\texttt{(p,3)} \wedge \texttt{dim}_2\texttt{(p,4)}$.

4.2 Memory

Human:	"Build a 3 by 4 Foo."
Parse:	$\texttt{Foo(p)} \wedge \texttt{dim}_1\texttt{(p,3)} \wedge \texttt{dim}_2\texttt{(p,4)}$
Query:	$\mathit{Define}(\texttt{Foo})$
System:	"Sorry, I don't know what 'Foo' is. Could you describe how to build the Foo for me using rows, columns, squares, and rectangles?"
Human:	"Build a square of size 3. Then, construct a row of 4 blocks above the square. The left end of the row should be above the upper right corner of the square."
Lift:	$\texttt{Foo(p)} \wedge \texttt{dim}_1(\texttt{p}, ?\texttt{d}_1) \wedge \texttt{dim}_2(\texttt{p}, ?\texttt{d}_2) \rightarrow$ $\texttt{square(a)} \wedge \texttt{size}(\texttt{a}, ?\texttt{d}_1) \wedge$ $\texttt{row(b)} \wedge \texttt{width}(\texttt{b}, ?\texttt{d}_2)\ \wedge \texttt{spatial-rel}(\ldots$

Figure 4: A dialogue triggered by the unknown word "Foo" for the configuration in Figure 1. LC issues a query that prompts a request to define "Foo" in terms of known shapes. This definition is first parsed and grounded as in Figure 3, then learned by being lifted and stored in M ($?\mathtt{d_k}$ identifies dimension variables).

A key challenge for communicating agents is the necessity to learn the interpretation of new concepts (e.g. names of unknown shapes, spatial relations, or actions)[1] that arise during the communication with the human. In our agent, the Memory module stores lifted representations of these acquired concepts that are parameterized for the configuration's dimensions, and hence generalize beyond the specific instances encountered by the agent. When the agent receives an unknown goal, it asks the human for a definition of the unknown concept and expects a new goal description $\mathcal{G}^N$ that defines it in terms of known concepts. This definition is then added to the agent's domain knowledge stored in M.

Learning a new concept is done via a "lifting" process, as follows:

1. When an unknown goal description $\mathcal{G}^U$ is received, LC issues a query Q^U which is then realized and posed to the human by LP.
2. LC receives a natural language response containing a new goal description $\mathcal{G}^N$. If $\mathcal{G}^N$ contains any unknown concepts, the previous step must be called recursively. Once $\mathcal{G}^N$ is complete, it is grounded and passed to PS.
3. If a successful plan was generated for $\mathcal{G}^N$, the concept declaration and goal definition are lifted by converting the given dimension and relative location constants to variables while ensuring parameter sharing between the new concept and its definition. A mapping is created and inserted into M. Lifting ensures generalization over arbitrary examples of the new learned concept.

Figure 4 illustrates an example of a description that elicits a query from the system for further information and the subsequent resolution process. Our present implementation of lifting is restrictive. The challenges in handling the general learn-

---

[1] We currently restrict ourselves to unknown shapes.

ing setting and potential principled approaches will be discussed in later sections.

4.3 Problem Solving

The problem solving module reasons about the configurations communicated to it as conjunctive logical expressions and generates a set of problem solving steps (plan) to achieve the given target configuration as the output. The problem solving module proceeds in an anytime fashion and generates a partial plan as far as possible. We employ a Hierarchical Task Network (HTN) planner (Erol et al., 1994; Nau et al., 2003; Ghallab et al., 2004) that searches for plans in a space of task decompositions. HTN allows for reasoning over different levels of abstraction, tasks and sub-tasks as well as primitive actions and is more intuitive for humans.

4.4 Dialogue Mediation

The role of the dialogue mediator in our framework is to guide the interaction with the human and delegate the tasks of parsing, planning, and query realization to the LC, PS, and LP modules respectively. The DM interacts with a GUI framework that allows for back-and-forth textual interaction between the human and the system as well as a visualization component that displays the output of the problem solver as block configurations on a grid. Via this framework, the DM accepts user-described goal descriptions and prompts the user to reword their utterances, clarify missing information, and define new shapes as needed.

The DM is also responsible for keeping track of the cumulative information gained about a goal configuration over a dialogue sequence in order to backtrack the states of the semantic parsing and problem solving components if mistakes occur during the interaction. In the semantic parser, backtracking consists of restoring the hidden states of the parser that were seen at a particular time of the interaction before the mistake was made. Backtracking in the problem solver involves deleting or modifying items in its goal description $\mathcal{G}$. But, since the parser conditions hidden states on all previously seen sentences, the DM is not able to selectively delete or replace information at arbitrary points in the timeline of the dialogue. Hence, our experiments process goal descriptions on a sentence-by-sentence basis, allowing for clarifying questions to be made and resolved per-sentence.

Our current agent uses a rule-based dialogue mediator (implemented as a finite state machine), which alternates between four sets of states:

1. **Goal description parsing and planning.** Given an input user description of a goal configuration, DM passes this input through the LC and PS pipeline, backtracking and requesting a rewording if either module encounters a failure.

2. **Querying for and resolving clarifications.** When LC returns a a query Q asking for values of missing dimensional or spatial features, DM requests that information from the user via the LP module (e.g., "What is the width of the row?"). The user may respond with a well-formed sentence describing the missing feature value ("The width of the row is 4") or with a fragment containing the desired information ("It's 4," "4," "Width is 4"). Given the context of the original query Q, DM extracts the value via a heuristic and reforms the user input into a well formed sentence, then returns to State 1 to handle the updated description.

3. **New shape learning.** When LC returns a query Q indicating an unknown shape or concept, DM requests the user to describe the desired configuration using known shapes (i.e., rows, columns, squares, and rectangles). The description handling process proceeds regularly as in State 1 until the user indicates they are finished defining their new concept. This triggers the learning process to lift and store the definition in M; the new concept can then immediately be used like any known concept in future user descriptions.

4. **Shape verification.** Every time the plan for an individual shape in a configuration has been resolved, DM outputs the plan to the visualization component and asks the user if the configuration up until that point in the dialogue is correct. If the user indicates that something is wrong, DM removes the entire shape from the history of the parsing and planning components and asks the user to retry describing the shape from scratch. Once the shape has been verified, however, no further modifications may be made to that shape.

# Shapes	Task 1	Task 2
1	100%	100%
2	64%	52%

Table 1: Accuracies for Task 1 and Task 2.

For our experiments, a simplified version of this dialogue mediator was used to feed examples through the system pipeline. More details can be found in Section 5.

4.5 Language Generation

Our current system uses a predominantly template-based approach to generate user queries. For clarification questions, we use a simple rule-based approach for clarifying questions that increases the level of question explicitness if the user continually fails to understand the question or respond in an appropriate manner. For example, if the query Q requests the width of a particular row, clarification questions may range from the very contextual “What’s the width?” to the fully explicit “What is the width of the row you are currently building?”

5 Methodology and Experiments

Although we ultimately wish to evaluate our agent’s live interactions with real human users, our current experiments feed it synthetically generated descriptions. For the descriptions with missing information, we randomly choose dimensional and/or spatial information to omit for every shape, providing the missing information in a follow-up response upon system query. We also generate sentences with descriptions with unknown terms that have to be clarified in further interactions. After a full interaction has been carried out (or halted early due to system comprehension failure), we automatically verify the system’s output configuration against the gold configuration.

Input consisting of sentences describing various block configurations and their corresponding logical forms was produced using a template-based generation system. A subset of this data was used for training the semantic parser; the rest was used for the following experiments. The generation system was set up to create configurations randomizing over shape types, sizes, and relative locations, and the templates for the sentences were designed to introduce lexical and grammatical variety. Overall, the generation system is able to produce tens of thousands of different configuration descriptions.

Human:	Build a square of size 3. Then, construct a row of 4 blocks above the square. The left end of the row should be above the upper right corner of the square.
System:	Is this okay so far? (y/n)
Human:	yes
System:	Great!

Figure 5: Task 1 (Complete descriptions)

Generated descriptions contain one sentence (for single shapes) and up to three sentences (for two shapes). The sentences have an average length of approximately ten words. Multi-sentence descriptions pose special challenges for semantic parsing due to the need to resolve coreference across sentences. Thus, increasing the number of shapes in a configuration dramatically increases the complexity of the parsing task.

The COG system was evaluated on three separate tasks as outlined in Section 2; we present the results below.

5.1 Task 1: Complete Descriptions

In the first part of the experiment, we test COG’s ability to handle complete descriptions of configurations (see Figure 5 for an example dialogue). The parameter that we change in the course of this experiment is the number of primitive shapes used to build the configuration, which we vary from 1 to 2. For each parameter setting we have 50 test instances; this totals 100 test instances of complete descriptions with no missing information. COG’s accuracy in interpreting these descriptions is shown in the left column of Table 1. The drop in accuracy in moving from one to two shapes is due to the drop in performance of the semantic parser when dealing with complex descriptions.

5.2 Task 2: Missing Information

In this experiment, we test COG’s ability to process descriptions with missing information (see Figure 6 for an example dialogue). The system handles these situations by asking clarification questions, obtaining responses and building the configuration by incorporating these subsequent responses. Again, we vary the number of primitive shapes in the configuration from 1 to 2 and have 25 test instances for each case. Further, each primitive shape could potentially have either or both the dimension and relative spatial position missing in

Human: Form a column.
System: What is its height?
Human: 5 blocks
System: Is this okay so far? (y/n)
Human: yes
System: Cool! Please continue to describe your configuration.
Human: Finally, form a vertical line of size 5 to the right of the column.
System: Can you describe the second column's location relative to the first?
Human: Ensure that the top end of the vertical line is next to the third row of the column.
System: Is this okay so far? (y/n)

Figure 6: Task 2 (Missing information)

Human: Construct a balloon of height 3 and width 4.
System: Sorry, I don't know what 'balloon' is. Can you describe how to build the balloon for me using rows, columns, squares, and rectangles?
Human: Build a column of height 3. Then, build a 4 by 4 square above the column. Ensure that the second column of the square is above the top end of the column.
System: Is this okay so far? (y/n)
Human: yes
System: Great! I've now learned what a balloon is.

. . .

Human: Build a balloon of height 2 and width 7.
System: Is this okay so far? (y/n)

Figure 7: Task 3 (Learning new shapes)

the initial description. For example, in a configuration involving 2 shapes, there could potentially be 3 pieces of information missing. The 25 test instances are generated such that the number of missing information pieces in a single instance is uniformly random in $[1, 3]$. COG's accuracy in interpreting incomplete descriptions is shown in the right column of Table 1. As before, the drop in accuracy in moving from one to two shapes is due to the drop in performance of the semantic parser when dealing with complex descriptions. The accuracy is further impacted by the interaction, there are more opportunities for the parser to misinterpret a natural language utterance.

5.3 Task 3: Learning New Shapes

In the final experiment, we evaluate COG's ability to learn new shapes (see Figure 7 for an example dialogue). We teach it five descriptions of new shapes, the results of which are presented in Table 2. Each new shape is defined using descriptions consisting of primitives (e.g., rows, columns, squares, and rectangles) of certain dimensions. We then tested the ability of the system to generalize and build other instances of the new shapes with altered dimensions. The "Outcome" column indicates whether the system was able to correctly lift all of the parameters of the input shape. Note that, when the dimension of a new shape consists of the sum of two dimensions (as in the third and fifth examples), parameter lifting fails for that dimension.

6 Discussion and Future Work

The goal of this paper was to present and study a new agent architecture that supports natural, symmetric communication with humans in the context of performing a real world task. Even in the simulated environment considered here, we had to address a number of challenges. First, language understanding to support task execution differs from e.g. standard semantic parsing in that it requires grounded meaning representations that can be executed and whose grounding may depend on the particular situation. Second, agents needs to be able to identify, request, and incorporate missing information that prevents task execution. Finally, agents need to be able to identify and learn the interpretation of new terms introduced during the interaction with the user.

Our current implementation has a number of obvious shortcomings. A primary bottleneck of COG is the semantic parser. If the input descriptions are not parsed correctly, there is very little the system can do to recover beyond asking for a rephrasing from the human. This creates problems when we attempt to experiment with complex configurations that involve three or more shapes. In these cases, the parser struggles to correctly identify the spatial relations for the third shape. Future work needs to address how to better handle longer and more complex descriptions.

Additionally, as the performance of Task 2 indicates, the dialogue mediator sometimes has problems translating partial responses into complete sentences that the parser can handle robustly. More work is required to develop a better treatment of implicit arguments, coreference, and discourse referents that are commonly present in these types of responses.

The generation component is also limited. For example, we currently cannot produce instructions for new shapes. We have also de-

Initial Descriptions	Further Explanations	Outcomes
Box of width 5 and height 3	Construct a rectangle of width 5 blocks and height 3 blocks.	All parameters lifted correctly
Balloon of width 4 and height 3	Build a column of height 3. Then, build a 4 by 4 square above the column. Ensure that the second column of the square is above the top end of the column.	All parameters lifted correctly
Balloon of width 4 and height 7	Build a column of height 3. Then, build a 4 by 4 square above the column. Ensure that the second column of the square is above the top end of the column.	Failed to lift height of balloon
L of height 6 and width 3	Build a column of 6 blocks. Then, build a row of 3 blocks to the right of the column. Make sure that the left end of the row is to the right of the bottom end of the column.	All parameters lifted correctly
L of height 6 and width 4	Build a column of 6 blocks. Then, build a row of 3 blocks to the right of the column. Make sure that the left end of the row is to the right of the bottom end of the column.	Failed to lift width of L

Table 2: Task 3 results for learning new shapes. We initially provide descriptions containing new shape terms with their parameters (left column). When prompted for further clarification, we then provide the descriptions in the middle column. After the shapes were learned, we instructed COG to build more instances of these new shapes while varying the size parameters to test how well the lifting worked; the result of this is in the right column.

veloped a grammar-based realizer inspired by OpenCCG (White and Baldridge, 2003; White, 2006) that operates over the first-order semantic representations used by our agent. We plan to augment the realizer's semantic lexicon with the learned definitions of predicates for new shapes, allowing our system to generate natural language instructions describing the new configurations.

One of the key challenges of the scenario we envision (and a fundamental problem in language acquisition) is the necessity to generalize across situations. This is required in order to learn general concepts from a few specific instances. At this point, our agent is able to generalize from a single example, but our learning mechanism is rather naive, and we can only handle simple parameter lifting from the primitive components of the new shape. To illustrate this, consider an example where we want to teach the system the shape "balloon". To do so, we must specify the dimensions of the square and column separately; consequently, the system will learn that the height of a "balloon" corresponds only to the height of the column (see Table 2). However, ideally we would like the system to learn a parameterization of "balloon" where the balloon's height corresponds to the sum of the square and column heights.

Our next step will focus on improving this mechanism. One possible direction involves the use of Inductive Logic Programming to induce high-level hypotheses from observations and background domain knowledge. One key challenge here, beyond learning, is a way to incorporate more complex learned hypotheses into our architecture in such a way that other components in our system can make use of it. A second challenge is that the agent is learning from a human that has limited knowledge, and little patience, so we cannot expect to see a large number of examples. We will consider the use of probabilistic logic models which can handle both issues by explicitly including the trade-off in the optimization function (Odom et al., 2015).

A final challenge is the application of our agent to new domains. Currently, the memory module contains all the knowledge required to plan and produce comprehensible responses. This declarative approach should generalize well to some simple enough domains, but will need to be extended to deal with more involved tasks and domains.

Acknowledgements

This work was supported by Contract W911NF-15-1-0461 with the US Defense Advanced Research Projects Agency (DARPA) Communicating with Computers Program and the Army Research Office (ARO). Approved for Public Release, Distribution Unlimited. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

References

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473* .

J. Clarke, D. Goldwasser, M. Chang, and D. Roth. 2010. Driving semantic parsing from the world's response. In *Proc. of the Conference on Computational Natural Language Learning (CoNLL)*. http://cogcomp.cs.illinois.edu/papers/CGCR10.pdf.

Li Dong and Mirella Lapata. 2016. Language to logical form with neural attention. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Berlin, Germany, pages 33–43. http://www.aclweb.org/anthology/P16-1004.

Kutluhan Erol, James Hendler, and Dana S Nau. 1994. HTN planning: complexity and expressivity. In *Proceedings of the Twelfth AAAI National Conference on Artificial Intelligence*. pages 1123–1128.

R. E. Fikes and N. Nilsson. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2(3-4):189–208.

Malik Ghallab, Dana Nau, and Paolo Traverso. 2004. *Automated planning: theory & practice*. Elsevier.

Jiatao Gu, Zhengdong Lu, Hang Li, and Victor O.K. Li. 2016. Incorporating copying mechanism in sequence-to-sequence learning. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Berlin, Germany, pages 1631–1640. http://www.aclweb.org/anthology/P16-1154.

Caglar Gulcehre, Sungjin Ahn, Ramesh Nallapati, Bowen Zhou, and Yoshua Bengio. 2016. Pointing the unknown words. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Berlin, Germany, pages 140–149. http://www.aclweb.org/anthology/P16-1014.

Robin Jia and Percy Liang. 2016. Data recombination for neural semantic parsing. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Berlin, Germany, pages 12–22. http://www.aclweb.org/anthology/P16-1002.

Dana S Nau, Tsz-Chiu Au, Okhtay Ilghami, Ugur Kuter, J William Murdock, Dan Wu, and Fusun Yaman. 2003. SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research* 20:379–404.

P. Odom, T. Khot, R. Porter, and S. Natarajan. 2015. Knowledge-based probabilistic logic learning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*. pages 3564–3570.

Stefanie Tellex, Thomas Kollar, Steven Dickerson, Matthew R. Walter, Ashis Gopal Banerjee, Seth J. Teller, and Nicholas Roy. 2011. Understanding natural language commands for robotic navigation and mobile manipulation. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*.

Michael White. 2006. Efficient realization of coordinate structures in Combinatory Categorial Grammar. *Research on Language & Computation* 4(1):39–75.

Michael White and Jason Baldridge. 2003. Adapting chart realization to CCG. In *Proceedings of the 9th European Workshop on Natural Language Generation*. pages 119–126.

J. M. Zelle and R. J. Mooney. 1996. Learning to parse database queries using inductive logic proramming. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*. pages 1050–1055.

L. Zettlemoyer and M. Collins. 2005. Learning to map sentences to logical form: Structured classification with probabilistic categorial grammars. In *Proceedings of Uncertainty in Artificial Intelligence (UAI)*.