Towards a Meaningful Natural Language Interface

Huda Khayrallah\textsuperscript{1,2}, Sean Trott\textsuperscript{1}, Luca Gilardi\textsuperscript{1}, Jerome Feldman\textsuperscript{1,2}

\textsuperscript{1}International Computer Science Institute, Berkeley, California, USA
\textsuperscript{2}Division of Computer Science, University of California, Berkeley, USA
\{huda, seantrott, lucag, feldman\}@icsi.berkeley.edu

Abstract

We present a system that allows for neurally based natural language communication with a robot simulator. This natural language interface is based on the Embodied Construction Grammar, part of the Neural Theory of Language project. As such, it provides an embodied basis for language analysis. This system is flexible with respect to both input language and output task, demonstrating the potential range of application.

Introduction

For innovations in robotics to have widespread reach, there must be an interface through which robots operators can control robots without programming. One way for humans to communicate is through natural language. The same principles apply outside robotics; natural language interfaces, such as Google Now and Siri, have become the expectation in smart phones.

We describe a system that can follow instructions and synthesize actions and procedures in natural language. (Oliva et al 2012)

Related Work

This work lies at the intersection of two areas, natural language interfaces—specifically for robotics—and the embodied construction grammar. Extensive work has been done on both ends.

Natural Language Interfaces

Natural language interfaces have been attempted for the last 50 years. SHRDLU, as shown in Figure 1, was described in Terry Winograd’s 1971 doctoral thesis, as an early example of a program that allowed the user to carry on a conversation, manipulate objects, and ask about the state of the block world. His thesis describes a present world where computers can be interacted with uniquely through computer languages, and a future where one could communicate with a computer in domain specific natural language (Winograd 1971). When describing the failures of machine translation he writes:

When a human reader sees a sentence, he uses knowledge to understand it. This includes not only grammar, but also his knowledge about words, the context of the sentence, and most important, his knowledge about the subject matter. A computer program supplied with only a grammar for manipulating the syntax of language could not produce a translation of reasonable quality.

This highlights the difference between manipulating language and understanding it. Much work has approached this problem by focusing on the manipulation of language for a purpose, in place of the understanding. For many contexts, this may be sufficient; we take a different approach.

Figure 1: SHRDLU

Embodied Construction Grammar

This work is powered by the Embodied Construction Grammar (ECG), and builds on decades of work of the Neural Theory of Language (NTL) project. The ECG represents meaning independent of specific linguistic constructions, and instead focuses on representing neural-conceptual structures (Feldman, Dodge, and Bryant 2009).
ECG is designed as:

1) A descriptive formalism for linguistic analysis
2) A computational formalism for implementing and testing grammars
3) A computational module for applied language tasks
4) A cognitive description for reduction and consequent experiments
5) A foundation for theories and models of language acquisition

In this work, we focus on principle 3; we are using ECG as the computation model for the natural language interface to a robot simulator. ECG has previously been demonstrated as a ‘computational module for applied language tasks’ for understanding solitaire card game instructions (Oliva et al. 2012). This work expands the system to the domain of simulated robotics. The other principles have been demonstrated in prior work (Chang 2008) (Mok 2008).

ECG—as a Natural Language Understanding (NLU) project—differs from Natural Language Processing (NLP) systems in several ways that make it good for our purpose. Basic NLP systems need to be trained on labeled data. This system does not have that requirement; however it does depend on a curated grammar, knowledge base, and vocabulary. Once these are created, they can be used and extended across systems.

System Architecture

As shown in the system diagram, in Figure 2, this system is designed to be modular; a crucial part of the design is that the ECG grammar is designed to work for any system.

The main programs are the analyzer, the specializer, the problem solver, and the robot simulator. The analyzer parses the user input with the ECG grammar and ontology and outputs a data structure called the SemSpec. This is used by the specializer to capture the task relevant information, which it sends to the problem solver in the n-tuple data structure. The problem solver then uses the information from n-tuple, along with the problem solver’s internal model of the world, to make decisions about the world and carry out actions. The user can observe the changes in the world. Additionally, the problem solver updates its model of the world after each action, so it can continue to make informed decisions and actions.

While this work focuses on English, the system also works in Spanish. The same analyzer, problem solver, and simulator can be used without alteration. The analyzer simply references a different grammar. Spanish and English have more than cosmetic differences, therefore use different constructions, hence a modified specializer is needed. The specializer then extracts the relevant information for the solver, and creates n-tuple in the same format as in English. This allows the problem solver and robot simulator to remain unchanged.

Discussion

<table>
<thead>
<tr>
<th>Supported Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) Robot1, move North then return!</td>
</tr>
<tr>
<td>2) Robot1, move to the blue box!</td>
</tr>
<tr>
<td>3) Robot1, move to the big red box then move to the small red box!</td>
</tr>
<tr>
<td>4) Robot1, move to location 1 2!</td>
</tr>
<tr>
<td>5) Robot1, move to Box1!</td>
</tr>
<tr>
<td>6) Robot1, move to a red box!</td>
</tr>
<tr>
<td>7) Robot1, move to the small box!</td>
</tr>
<tr>
<td>8) Robot1, move to the big red box!</td>
</tr>
<tr>
<td>9) Robot1, move to a big box!</td>
</tr>
<tr>
<td>10) is Box1 red?</td>
</tr>
<tr>
<td>11) Robot1, if the small box is red, move to the big red box!</td>
</tr>
<tr>
<td>12) Robot1, if the small box is red, move to it!</td>
</tr>
<tr>
<td>13) Robot1, move to the small box!</td>
</tr>
<tr>
<td>14) is the small red box near the blue box?</td>
</tr>
<tr>
<td>15) Robot1, move behind the big red box!</td>
</tr>
</tbody>
</table>

Table 1: Sample supported input
Table 1 highlights a representative sample of working input, corresponding to the scene in Figure 3. There is an obvious focus on motion, due to the functionality of the robot used. The location to which the robot is instructed to move include specific locations “location 1 2,” and specific items “Box1.” The system can also handle more complicated descriptions, using color and size. Additionally, when asked for an indefinite object, such as, “a red box,” and there are multiple objects that fit the description, one of the red boxes is arbitrarily chosen.

In addition to moving, the system can also handle yes or no questions—as demonstrated in Example 10, in Table 1. Example 11 demonstrates a conditional; the robot will only perform the instruction if the statement is true. The system can also handle basic referent resolution, as demonstrated in examples 12 and 13. This is done by choosing the most recent antecedent of the proper kind. This method is demonstrated by Oliva et al, and is based on the way humans select antecedents.

**Error Handling**

<table>
<thead>
<tr>
<th>No Grammatical Parse</th>
</tr>
</thead>
<tbody>
<tr>
<td>Robot1, move to the white box! No parse found for: Robot1, box big red move to the white box!</td>
</tr>
<tr>
<td>Lack of Referent</td>
</tr>
<tr>
<td>Robot1, move to the big room! Problem: &lt;specified object&gt;</td>
</tr>
<tr>
<td>Robot1, move to the pink box! Problem: does not exist.</td>
</tr>
<tr>
<td>Under-Specified Input</td>
</tr>
<tr>
<td>Robot1, move to the red box! Problem: there is more than one object that matches that description. Please be more specific.</td>
</tr>
<tr>
<td>Robot1, move to the big box!</td>
</tr>
</tbody>
</table>

**Table 2: types of errors**

If the analyzer cannot parse the input, this causes an error. This may occur due to a word not being in the lexicon. For example, white has not been added for the existing demo, and will return this error. Additionally, even if the words themselves are found this error can occur if the input does not match a construction.

If the user attempts to describe an object that does not exist in the simulation, this leads to the error, as would be expected. Additionally, if there is more than one object that matches the description, and a definite article is used (e.g. “the”) an error also occurs.

**Extended example**

In order to demonstrate the integration and functionality of the system, we will trace an extended example from text to action. We will consider the command, “Robot1, if the small box is red, move to the big red box!” since it combines some of the advanced functionality of the system. This will be demonstrated in the context of the example world in Figure 3; the actions only make sense in that context. This paper focuses on the problem solver, though it is best understood in terms of its interaction with the rest of the system.

**Analyzer**

The input text is first parsed by the analyzer program. Through the ECG grammar, the analyzer uses syntactic and semantic properties to create a best-fit model and produce a SemSpec. This SemSpec is a grammatical parse of the sentence, according to the ECG grammar (Bryant 2008). The SemSpec for this example can be found in Appendix A.

**Specializer**

The specializer extracts the relevant information for the problem solver from the SemSpec. This output is in the form of an n-tuple, a data structure based on Python dictionaries. The n-tuple for this example is included in Table 3. When comparing the n-tuple to the SemSpec, the careful reader may notice some parallels. The task of the specializer is to extract the information relevant to the problem solver in a consistent and predictable way.

```python
Struct(parameters=[Struct(kind='conditional', condition=Struct(kind='query', action='be', acted_upon='{color': 'red', 'type': 'box', 'givenness': 'uniquelyIdentifiable', 'size': 'small'}, predication={'color': 'red'}), command=Struct(heading=None, action='move', control_state='ongoing', acted_upon='robot1_instance', distance=Struct(units='square', value=4), kind='execute', direction='KD', goal={'objectDescriptor': {'color': 'red', 'type': 'box', 'givenness': 'uniquelyIdentifiable', 'size': 'big'}})), return_type='error_descriptor', predicate_type='conditional')
```

**Table 3: n-tuple**

**Problem solver**

The problem solver parses the n-tuple to determine the action needed, and make that action. It begins by determining the predicate type to understand the request. In this case, it is a conditional. It then evaluates the condition, and, in order to do so, it determines the referent. In our working example, it must recognize it is looking for a box, and then searches for a box that matches the description, in this case big and red. The problem solver maintains a data structure that contains the current state of the world. This is updated after each command to the simulator, allowing the problem solver to have an understanding of the current world.

Once it has determined the box, it is then able to evaluate if this box is red. If so, it interprets the command, to determine that the robot is being asked to move, and then it must determine the location. Since the location is not specified by coordinates, the problem solver
disambiguates the location, by first finding the goal box, again by recognizing it is looking for a box, and then searching for a box that matches the description, in this case big and red. Once the location of the box is determined, the decision to move to that location is made. The call to move is then made to the wrapper class of the Robot simulator API, here MORSE. This additional level of abstraction allows the system to work with an arbitrary robot simulator (or even a physical robot) assuming it supports the same primitives.

**Simulator**

This system is built on top of MORSE (Echeverria et all), which in turn relies on Blender. MORSE is an open source (BSD-3 clauses) system level simulator designed for academic robotics. While our system is designed to work on an arbitrary simulator, it has been influenced by the specifications of this one. MORSE provides some useful functionality, including realistic physics and a variety of interfaces (including the Python bindings we use). It also has some key limitations, such as the lack of path planning. Blender is an open source application for 3-D modeling and rendering. The use of Blender allows for realistic physics simulations, and easy modeling. This work was primarily developed with Morse 1.2 and Blender 2.69; it has also been in limited use with MORSE 1.2.1 and Blender 2.71.

**Conclusion**

This paper demonstrates a fully integrated yet modular system that provides a natural language interface to a robotic simulator based on embodied semantics. In combination with Oliva et al, this demonstrates that the ECG and Analyzer can be used for diverse applications. The Spanish functionality demonstrates the flexibility of the problem solver. The use of the ECG allows for a deep semantic understanding, which allows for full understanding of different input languages, a solid framework for analyzing an embodied concept, such as spatial relations.

**Limitations**

The functionality is currently limited by the items in the lexicon and the constructions. These must be manually added. Additionally, each new construction must be handled in the Specializer.

**Future Work**

This project is still in active development. In order to demonstrate the functionality of the system in the simulation, basic obstacle avoidance or path planning is needed.

On the system level, the ontology referenced by Analyzer needs to be shared with the Problem Solver in order to give both modules the relevant information about the terms used.

Additionally, the ability to create a variable (ie. $x$ such that $x$ is a red box, and $x$ is near the blue box) define a series of commands as one task (ie. Visit means to go a location and then return) mak are under development.

**References**

Bryant, J. E. 2008. Best-Fit Constructional Analysis. Ph.D. diss., Department of Computer Science, University of California at Berkeley


Mok E., 2008. Ph.D. diss., Department of Computer Science, University of California, Berkeley, CA

Oliva J.; Feldman J.; Giraldi L. and Dodge E. Ontology Driven Contextual Reference Resolution in Embodied Construction Grammar. 2012. In the proceedings of the 7th Annual Constraint Solving and Language Processing Workshop. Orléans, France


**Acknowledgments**

This work is supported by the National Science Foundation Research Experiences for Undergraduates (REU) Site Program SUPERB-Information Technology for Sustainability award number 1359499 and the Office of Naval Research grant number N000141110416.